

54

Encore 153 numéros avant l´an 2000

MARS 1987

Le lièvre...

EDITORIAL

FORTH se porte de mieux en mieux. Après AMSTRAD et IBM, nous vous mijotons une surprise pour les semaines à venir coté ATARI. Ainsi, votre "travail" consiste maintenant à développer des programmes, des utilitaires, des outils en FORTH. Nous souhaiterions pouvoir proposer rapidement une disquette contenant ces outils.

Coté parution, nous ferons un GROS effort pour combler le retard. Mais vous qui vous intéressez à l'intelligence artificelle et aux systèmes experts, méditez cette phrase empreinte de récursivité:

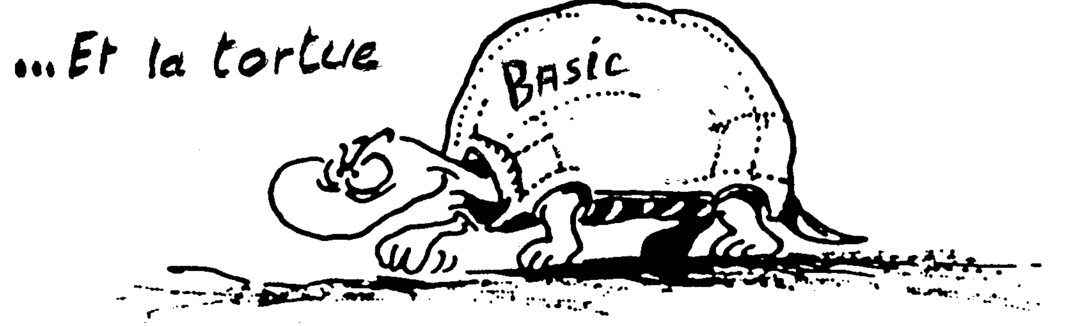
LOI DE HOFSTADER: tout travail prend toujours plus de temps que celui prévu pour son accomplissement même en tenant compte de la loi de HOFSTADER.

et qui est extraite du livre 'GODEL ESCHER BACH, les brins d'une guirlande éternelle' (éditions InterEditions PARIS), dont je recommande la lecture en remplacement des japoniaiseries et berlusconneries télévisuelles dont nous abreuvent abondamment les lucarnes du PAF (et là c'est trop facile, je ne ferai pas de jeu de mot).

SOMMAIRE

ASSOCIATION JEDI 17, rue de la Lancette 75012 PARIS

ou en nous téléphonant au: (1)43.40.96.53 permanence ou répondeur 24h24
(1)46.56.33.67 secrétaire aux heures de bureau.



QUESTIONS ET REPONSES
R2 R2 R5

REPONSE A LA QUESTION 2 : par C.LECLERC (91200 ATHIS MONS)
Voici les équivalents de INP et OUT du B3X en FORTH:

```
HEX
CREATE OUT ( port val ---)
D9 C,         EXX
C1 C,         POP BC
79 C,         LD  A,C
C1 C,         POP BC
ED C, 79 C,   OUT (C),A
D9 C,         EXX
DD C, E9 C,   JP  (IX)
SMUDGE

CREATE INP ( port --- val)
D9 C,         EXX
C1 C,         POP BC
ED C, 78 C,   IN  A,(C)
06 C, 00 C,   LD  C,A
C5 C,         PUSH BC
D9 C,         EXX
DD C, E9 C,   JP  (IX)
SMUDGE
```

Une bascule MAJ/min en FORTH:

```
HEX
: BASCULE   ( ---)
 FEB1 VC@  1  XOR  FEB1 VC!  ;
```

Hectoriennement vôtre!

REPONSE A LA QUESTION 2 : par CLUB HECTOR LIMOGES (87000 LIMOGES)
Cher Hectorien,
Veuillez trouver ci-joint les réponses à votre double question no2 parue dans la revue JEDI de janvier.

Nous nous permettons de vous adresser ce courrier en même temps qu'à la revue afin que vous puissiez en prendre plus rapidement connaissance!

1)Equivalences INP et OUT: il vous suffit de créer ces mots avec du code machine comme suit:

```
HEX CREATE OUT SMUDGE  ( oct add ---)
    E1 C, D1 C, C5 C, 4D C, 59ED , C1 C, E9DD ,
    CREATE INP SMUDGE ( add --- oct)
    E1 C, C5 C, 44 C, 4D C, 68ED , 0026 ,
    C1 C, E5 C, E9DD , DECIMAL
```

L'appel se fera sous la forme: (exemple) HEX 0 49 OUT DECIMAL

2)Bascule MAJUSCULE/minuscule dans un programme FORTH. Il y a lieu d'utiliser le multiflag d'interruptions situé à l'adresse &FEB1 en page vidéo et plus précisément mettre le bit 0 à 1 pour passer en MAJUSCULE et à 0 pour repasser en minuscules et ceci afin de ne pas altérer les autres fonctions de ce multiflag (poker 116 ou 117 pour min/MAJ obtient ce résultat mais détruit les autres bits s'ils ont été éventuellement utilisés). Les codes machines SET No de bit pointé par HL et RES (idem) qui forcent à 1 ou 0 le bit de l'octet concerné remplissant bien cette fonction nous vous proposons deux routines en code machine:

```
: MAJUSCULE [ HEX LATEST PFA DUP CFA !
   32 C, 0800 , 21 C, FEB1 , CB C, C6 C,
   32 C, 0808 , DD C, E9 C,  ] ;
: MINUSCULE [ HEX LATEST PFA DUP CFA !
   32 C, 0800 , 21 C, FEB1 , CB C, 86 C,
   32 C, 0808 , DD C, E9 C,  ] ;
```

Nous espérons avoir répondu assez clairement à vos questions. Nous nous tenons à votre disposition pour d'éventuelles précisions si nécessaires (il y a lieu de notre que nous avons fait paraître dans Megahector toute une étdude sur les interruptions du Z80 et les applications au HRX qui permet de faire exécuter plusieurs programmes en même temps!!!)

REPONSE A LA QUESTION 5: par M.PETREMANN (93160 NOISY LE GRAN
Il s'agit, dans la question de Mr ZUPAN, de ré-écrire facilement le source d'un mot code avec virgule et c-virgule.

Ce problème est certes intéressant, mais il semble moins évident de le résoudre que ne le pense Mr ZUPAN. Une solution n'expoitant que des valeurs litérales ne peut pas avoir d'utilité pratique pour la simple raison que les adresses d'implantation de certaines valeurs risquent de varier en fonction de ce qui a été précédemment compilé.

Pour ma part, cette idée m'a effleurée avant que M.ZUPAN ne la formule dans JEDI. Voici comment j'ai règlé ce problème sur l'assembleur 6809 pour THOMSON.

En premier lieu, on réécrit les mots c-virgule et virgule à partir de mots vectorisés:

```
DEFER C,
DEFER ,
```

Pour ceux qui disposent de l'assembleur FORTH 6809, ce travail est déjà fait. Il reste maintenant à définir deux routines réalisant les actions primitives de ces mots en y rajoutant une option d'affichage du code en cours de compilation:

```
: .C,
DUP BASE @ >R
HEX  CR
HERE 0 <# 58 HOLD # # # # #> TYPE
U. C, R> BASE ! ;

['] .C, ASSEMBLER IS C, FORTH

: .,
DUP BASE @ >R
HEX  CR
HERE 0 <# 58 HOLD # # # # #> TYPE
U.  , R> BASE ! ;

[']  ., ASSEMBLER IS  , FORTH
```

Pour la version 83-Standard CP/M ou MSDOS, il faudra recompiler l'assembleur en définissant au début de l'assembleur les mots c-virgule et virgule:

```
DEFER C,   DEFER ,
```

puis reprendre les définitions ci-dessus. Sortir de l'édition du fichier CPU8080.BLK (version CP/M), CPU8086.BLK (version MSDOS) et taper:

```
BYE
KERNEL EXTEND80.BLK   ou KERNEL EXTEND86.BLK
puis sous FORTH
START
```

On se propose ensuite d'assembler la définition suivante pour THOMSON, dont l'exécution délivre l'adresse de l'octet contenant le point situé aux coordonnées graphiques x y:

```
HEX 4000 CONSTANT ECRAN DECIMAL
     40 CONSTANT SCRLAR

CODE CALC ( x y --- adr)
x pshs  d pulu
1 ,u lda   ECRAN # ldx   a,x leax
SCRLAR # lda   aslb aslb aslb  mul
d,x leax   0 ,u stx   x puls
rts   HERE>CFA NEXT END-CODE
```

L'opération n LOAD, n étant le numéro de bloc contenant le source ci-dessus, affiche octet par octet le code compilé ceci par exécution des mots virgule et c-virgule en version vectorisée:

```
8400:34
8401:10
8402:37
8403:6
8404:A6
  ...etc...
8417:39
8418:BD
8419:8400
841B:7E
```

Suite p.7

LES TRAITEMENTS DE TEXTE
OBJECTIFS ET MOYENS

par Marc PETREMANN

Dans les premiers numéros de JEDI, nous diffusions un programme en FORTH permettant d'effectuer du traitement de texte sur ORIC-1. Ce programme certes performant, était loin d'être complet si on le compare aux logiciels professionnels spécialisés dans ce genre de travail. Au cours de notre propos, nous passerons en revue toutes les considérations à prendre en compte pour faire le meilleur choix dans un domaine en évolution constante: l'édition et la composition électronique.

HISTORIQUE PERSONNEL

Mon premier contact avec un système capable d'effectuer du traitement de texte s'est produit lors de la composition du livre 'INTRODUCTION AU ZX-FORTH' (Ed Eyrolles). Cet ouvrage a été intégralement typographié par les auteurs sur une machine à écrire RANK XEROX 620 (vous savez, celle qui, dans la pub, a une touche de magie) A l'usage, ce système s'est révélé très rigide. Sa capacité standard est limitée à quelques pages seulement. Il permettait néanmoins la justification à droite et toutes les fonctionnalités élémentaires que l'on est en droit d'attendre d'un traitement de texte. Petit avantage, une coupure de courant accidentelle n'avait que peu d'incidence sur le texte en cours de composition, la machine XEROX 620 sauvegardant sa mémoire pendant 2 à 3 mois.

Las de ne pouvoir reprendre le contenu d'un chapitre en entier pour correction éventuelle, nous nous sommes équipé pour le second ouvrage 'TOURS DE FORTH' d'un système à moins coûteux et plus performant, une BROTHER CE70 avec interface parrallèle/série et lecteur de micro-disque. Cette configuration pouvait être considérée comme une machine à écrire ordinaire ou comme imprimante, option dont nous ne nous sommes pas privés d'exploiter. Le résultat de la maitrise complète de cette machine a permis notamment la typographie intégrale de l'ouvrage 'PROGRAMMER EN LOGIQUE AVEC MICRO-PROLOG'. Pour ceux qui paraissent étonné d'apprendre que ce sont les auteurs eux-même qui typographient les ouvrages, je leur signale que beaucoup d'éditeurs sous-traitent cette tâche. Il n'est donc pas inintéressant pour l'auteur d'assumer ce travail simultanément à celui de création ou de traduction. Dans le cas du premier ouvrage cité, la typographie à rapporté à ce jour plus que les droits d'auteur!!

Dans le même temps, nous démarrions la revue JEDI. Je me suis donc équipé d'un système BROTHER EP44 et qui a servi à la composition de nombreux numéros. Cette machine, bien que légère en performances et en capacité mémoire, est dotée en standard d'un interface RS232 programmable en vitesse et protocole. Les piles assurent la sauvegarde des sélections. Le programme KERMIT du no 18 de JEDI a été listé en couplant la machine EP44 à un modem 300 bauds et avec pour seul interface de visualisation de transmission un écran LCD de 1 ligne de 8 caractères!

Par le suite, la machine EP44 a été couplée au TO7 via l'interface série, permettant le transfert des textes composés sur THOMSON.

Et maintenant, JEDI est composé à l'aide de WORDSTAR sur système BONDWELL. Pour information, le système BONDWELL II est un portable avec écran LCD 25 lignes, 80 caractères fonctionnant sur batterie. Un lecteur disquette 3'1/2 complémentaire peut être connecté; le hasard veut que le lecteur SPECTRAVIDEO MSX fonctionne très bien sur le BONDWELL II sans adaptation aucune, ce qui porte la capacité disque du BONDWELL II à 720K.

A l'usage, WORDSTAR se révèle tellement pratique que tous les textes sont saisis et composés à l'aide de ce programme, qu'il s'agisse de textes documents ou de programmes source: dBASEII, PASCAL, Assembleur, BASIC, etc... La frappe en deux colonnes par page c'est également WORDSTAR.

LES OBJECTIFS D'UN TRAITEMENT DE TEXTE

Pour choisir un bon traitement de texte, il faut en premier lieu connaître parfaitement l'usage que l'on en aura. En effet, il est inutile de se lancer dans l'acquisition d'un équipement coûteux s'il s'agit de composer quelques lettres. Par contre, la composition de documents de type professionnels exige une qualité et des performances impossibles à réaliser à l'aide de matériel 'bas de gamme'.

Considérons pour un premier exemple, un usage très ponctuel, la composition de documents et courriers. Ce type de travail peut être assumé sans difficulté par un système autonome: machine à écrire à mémoire. On en trouve à des prix très abordables (BROTHER EP44, CANON, etc...) dans une échelle de prix comprise entre 1400 et 4000 Fr.

Pour second exemple, les textes à traiter sont des brouillons destinés à être remaniés fréquemment, mais ne contenant que du texte. Pour celà, un ordinateur devient indispensable. Les textes peuvent être des documents ou des listings. Le traitement de texte doit par conséquent savoir traiter différemment ces deux types de textes. Soyez exigeants avec les fonctionnalités du traitement de texte:

- gestion par blocs
- fusion de texte
- justification par bloc
- recherche, insertion, suppression de mots ou groupes de mots

Les options complémentaires ne seront pas à négliger:

- passage de paramètres
- contrôle de la pagination
- contrôle de la forme (gras, souligné, interlignage..)
- césure automatique sélectable
- correction automatique sélectable

Pour dernier exemple, l'utilisateur désire intégrer des schémas, dessins, photos digitalisées. En excluant les systèmes typographiques spécialisés (WANG pour exemple), des programmes 'hauts de gamme' permettent ce genre de travail sur des micro-ordinateurs: PAGE-MAKER sur Mac INTOSH, PERSONAL PUBLISHER sur IBM. Dans une moindre mesure certains systèmes, tel THOMSON TO9, sont également équipés de traitement de texte permettant le mixage texte/graphisme. Inutile de préciser que les prix des logiciels de composition représentent un investissement non négligeable.

LES MOYENS A METTRE EN OEUVRE

Soyons objectifs, si vous disposez d'une machine à écrire et d'un ordinateur sans traitement de texte, il est inutile d'en acquérir un pour taper épisodiquement une lettre de 10 à 20 lignes. Passez-vous d'une acquisition que vous n'amortirez jamais. Si vous savez programmer, à la rigueur, vous pouvez créer votre propre traitement de texte. Si vous l'écrivez en BASIC, vous êtes très courageux:

- vous devez pouvoir entrer du texte 'au kilomètre'.

- après saisie, on doit pouvoir se positionner à n'importe quel endroit du texte.

- en pointant en un endroit quelconque du texte, on doit pouvoir surimprimer, détruire ou insérer des caractères, une ligne ou un bloc de caractères (plusieurs lignes).

Arrivé à ce point, en BASIC pour les personnes très courageuses, il faut pouvoir aussi disposer de:

- la sauvegarde et chargement sans altération.

- impression avec/sans justification de tout ou une partie du texte.

- découpage en pages avec ou sans numérotation.

Après çà, si vous n'êtes pas découragés et qu'il vous reste encore un peu de place en mémoire, achevez votre création en disposant des fonctions de:

- mise en gras, souligné, italique, etc.. et toutes options typographiques dont dispose votre imprimante.

Pour les autres, ceux qui diposent ou envisagent l'acquisition d'un traitement de texte, il faut considérer sur quel système il est destiné à fonctionner: →

- un modèle dit 'familial' est à exclure d'office surtout si l'on ne dispose pas d'au moins un lecteur de disque. La capacité mémoire doit être confortable, un minimum de 48K est requis pour pouvoir héberger à la fois le traitement de texte et le texte. Le système d'exploitation doit être compatible avec une large gamme de matériel. Les machines trop exotiques sont à proscrire, même si une démonstration époustouflante laisse loin derrière les autres programmes de traitement de texte: le modèle XWRTPRT de MACHIN ne sera peut être plus fourni par le constructeur dans 6 mois, le vendeur en faillite (si, si, çà existe...) après avoir vendu 10 exemplaires seulement de son produit 'haut de gamme, haute performance, petit prix'. Je n'hésiterai pas ici à citer le cas du NEWBRAIN, une machine très performante et dont l'importateur a encaissé des sommes considérables pour des lecteurs disques dont les clients attendent toujours la livraison. J'ai eu l'occasion, à l'époque, d'essayer leur 'traitement de texte (700 Fr TTC)' vendu sans documentation et sur cassette. Le traitement de texte en question était écrit en BASIC et faisait 3k de capacité mémoire. En deux mots: une escroquerie. Heureusement, ce programme a eu le temps de faire des petits, histoire de rentrer dans les frais (pour les puristes s'interdisant de faire de la piraterie, je signale que l'on ne peut porter tort à un circuit commercial défaillant, il y a des nuances à apporter à la notion de copie pirate...).

A signaler dans l'affaire NEWBRAIN, les efforts méritoires de l'association PARIS-MICRO pour assurer la survie de ce système en y adaptant un lecteur disque au format OSBORNE et le CP/M, ce qui a permis d'entretenir ensuite de bon rapports entre PARIS-MICRO et OUF.

Votre système doit disposer des caractères accentués. Si ce n'est pas le cas, vérifiez si vous disposez d'un utilitaire (SETUP sur AMSTRAD par exemple) permettant de transcoder votre clavier (pavé numérique transformé en clavier délivrant les caractères accentués).

Tous les caractères visualisés sur l'écran doivent correspondre à ceux imprimés. Sinon, vérifier si votre traitement de texte dispose d'un utilitaire de transcodage. Si ce n'est le cas, vérifiez si votre système d'exploitation dispose d'un 'handler' d'impression ou s'il dispose d'un utilitaire de transcodage (toujours SETUP sur AMSTRAD).

Après traitement, votre texte doit pouvoir être imprimé. Pour ce faire, prenez en considération les points suivants:

Vérifiez si votre imprimante est parfaitement compatible avec votre système. Si la majorité des traitement de texte transmettent en format 7 bits, il n'est pas exclu de devoir disposer du format 8 bits. Ici encore, la liaison parrallèle ou série est délicate. Si par exemple vous disposez d'un système AMSTRAD CPC, faites transformer votre système pour qu'il sorte les caractères sur 8 bits en parrallèle. Si vous disposez en plus d'un interface série, vous pourrez transmettre vos textes en format ASCII vers des systèmes différents (imprimante série, modem, autre système de traitement de texte...).

Si vous avez des fichiers de grande capacité à éditer, choisissez de préférence une imprimante disposant d'un tampon. Si ce n'est le cas, vous pouvez utiliser un utilitaire simulant un tampon d'impression (DSPOOL sur IBM et compatibles) ou acquérir un tampon d'impression (voir le catalogue de la Sté INMAC, ZI PARIS-NORD). En effet, il n'y a rien de plus frustrant d'attendre plusieurs minutes, voire plusieurs heures (100 pages sur SEIKOSHA ou MANNESMAN en qualité courrier) pour pouvoir reprendre la main sur votre système.

Quoi qu'on puisse dire, aucune imprimante matricielle ne peut concurrencer la qualité d'une imprimante à marguerite. Le choix douloureux entre imprimante à aiguilles ou à marguerite peut être tranché en acquérant simultanément une imprimante à aiguille et une imprimante à marguerite. En effet, pour le prix d'une imprimante à aiguille 'haut de gamme' vous pouvez disposer de deux imprimantes:

- l'imprimante matricielle pour tirer les listings et brouillons de texte. L'option d'impression graphique permettra l'impression de graphiques et dessins (cas de la SEIKOSHA SP1000, MT80, EPSON, Apple IMAGEWRITER...). Le prix des consommables et la vitesse d'impression amortissent rapidement une machine en utilisation intensive.

- l'imprimante à marguerite pour tirer des documents définitifs et de qualité. Un large choix de roues d'impression permet de personnaliser vos documents. Choisissez une machine dont le pas d'incrémentation horizontal et vertical peut être programmé. Ainsi, une imprimante de type QUME peut imprimer jusqu'à 80 lignes, 110 caractères par lignes, si elle est équipée d'une roue d'impression SQUARE 12/15 (pim=pas d'impression) en programmant le pim horizontal à 9 unités et le pim vertical à 7 unités. Certaines pages de JEDI sont composées de cette manière, ce qui évite de passer par la photoréduction. En option, une boîte de boules Quies, car les imprimantes à marguerites sont très bruyantes.

- l'imprimante à laser. Un investissement certes, mais de plus en plus abordable, alliant les avantages de l'impression matricielle (texte et graphisme) et la qualité de l'impression à marguerite. Un silence incomparable. En outre, dans le cas d'édition de documents en petite quantité, vous économiserez des frais de reproduction par photocopie. Pour exemple, l'imprimante CENTRONICS Laser a une capacité mémoire de 1Moctets, imprime à la vitesse de 8 copies minutes et peut être connectée à tout système en sortie parrallèle ou série (fiche AMPHENOL-CENTRONICS). Pour les documents recto/verso, il faudra bidouiller un peu les fichiers texte de manière à sortir une série de feuilles correspondant aux pages paires, puis retourner la série éditée pour les pages impaires.

Les performances de votre imprimantes doivent être en rapport avec celles de votre traitement de texte. Pour exemple, WORDSTAR ne travaille que sur des textes alphanumériques, alors que WORD autorise l'insertion de caractères semi-graphiques. Dans le premier cas, pratiquement n'importe quel type d'imprimante convient (matricielle, à marguerite, laser), alors que dans le second cas, elle doit être dédiée IBM et compatibles. Donc un plus qui restreint le choix du système et des moyens. A noter, pour WORDSTAR, que les commandes de ce traitement de texte ont été reprises en partie dans l'éditeur de Turbo-PASCAL et dans le traitement de texte PRACTITEXTE (éditeur FIL) pour THOMSON TO7/TO9 ou IBM.

Dans le cas de traitement de texte permettant le mixage texte et image, on ne peut prendre qu'une imprimante matricielle dédiée (image-Writer pour Mac-Intosh par exemple) ou Laser avec logiciel d'adaptation. Dans le cas du THOMSON TO9, ce choix se limite à l'imprimante du constructeur (une SEIKOSHA SP1000 rebaptisée avec EPROM modifiée pour THOMSON).

Pour ceux qui ne disposent pas de logiciel permettant le mélange texte-image, qu'ils se consolent, il suffit de réserver dans le texte la place pour leur dessin, puis à l'édition, rajouter le dessin par collage. La photocopie ou l'impression feront le reste. Comment croyez-vous que nous procédons pour JEDI. L'investissement se limite à une paire de ciseaux et de la colle. Pour les encadrés, je conseille le ROTRING (pas un logiciel, mais un stylo à encre de chine) et de supprimer les bavures avec du Tipex. Pour les gros titres, voir les catalogues LETRASET et MECANORMA. Ces techniques 'manuelles' restent valables même pour un ouvrage typographié destiné à l'édition professionnelle. Elle a l'avantage de résoudre tous les problèmes avec des moyens de fortune mais efficaces. Prenons le cas d'un ouvrage informatique, vous courrez moins de risque d'erreur si vous coupez et collez vos listings dans votre texte que si vous les recopiez.

LES SATELLITES DU TRAITEMENT DE TEXTE

En général, un traitement de texte peut se suffire à lui-même, que ce soit pour traiter des documents ou des programmes. Dans ce cadre d'activité, la maintenance d'un programme à l'aide d'un traitement de texte sera facilitée. Pour exemple, renommer une variable dans un programme sera exécutée sous WORDSTAR en activant la fonction ^QA (recherche et remplacement) de chaînes. Dans le cas de programmes très longs, le positionnement à un endroit quelconque sera effectué par la fonction ^QF. A noter que sous TURBO-PASCAL, vous pouvez disposer des mêmes fonctions que WORDSTAR, mais avec une capacité mémoire limitée. En effet, certains langages acceptent des fichiers de commande de taille quelconque (cas de dBASE II). WORDSTAR ou WORD seront donc appropriés au traitement des fichiers de commande dBASE. De même, vous pouvez traiter les fichiers source BASIC, FORTRAN, C, Assembleur, LOGO, PROLOG.

→

Mais un autre intérêt de WORDSTAR (ou WORD) est de permettre la fusion de fichiers. Pour exemple, prenons un petit programme PROLOG dont on aura vérifié le fonctionnement à l'aide de l'interpréteur PROLOG; on se propose, dans le cadre d'une notice utilisateur d'intégrer le listing de ce programme. Pour ce faire, il ne sera pas nécessaire de le retaper. Il suffira d'utiliser la commande suivante:

```
...texte... avec l'exemple suivant:

.FI programme.PRO

...suite du texte...
```

L'impression sera activée par l'option M pour MAILMERGE. Sur le papier, WORDSTAR insérera automatiquement le listing du programme à l'emplacement marqué par '.FI'.

De même, on peut insérer des données traitées par un fichier de commande dBASE:

```
SET ALTERNATE TO DOCUMENT.TXT
SET ALTERNATE ON
 ... traitement ...
SET ALTERNATE OFF
```

puis sous WORDSTAR:

"Cher client, voici le détail des opérations effectuées durant le dernier trimestre sur votre compte:

.FI DOCUMENT.TXT

Le montant est à nous faire parvenir sous quainzaine"

Pour exemple, un autre cas où l'interaction entre traitement de texte et SGBD (Système de Gestion de Base de Données) est fructueux, est l'affectation d'un fichier texte en tant que donnée. Ainsi, dans une base de données quelconque , disons un catalogue de produits industriels, il s'agit d'affecter un commentaire à une gamme de produits spécifiques. Mais ce commentaire peut faire dans certains cas deux lignes, dans d'autres cas trois pages. Or sous dBASE II, on ne peut gérer de champ de taille variable. Mais par contre, on peut entrer la référence du texte contenant le commentaire. Ainsi, à partir de trois fichiers texte contenant trois commentaires:

DOC1.TXT DOC2.TXT DOC3.TXT

on affectera dans un champ nommé arbitrairement DOCAFFER (pour document afférent), sous dBASE, une des trois références ci-dessus. A l'édition, le fichier de commande éditant les références d'un produit sera du style:

```
SET ALTERNATE TO FICHES.TXT
SET ALTERNATE ON
 ... traitement ...
? ".FI "+DOCAFFER

 ... suite traitement ...
SET ALTERNATE OFF
```

et en reprenant le contenu du fichier texte ainsi généré, FICHES.TXT en l'occurence, on y verra:

```
...texte... Les conditions d'achat de ce produit sont:

.FI DOC1.TXT

...suite texte...
```

Ce texte est à retraiter par MAILMERGE pour une édition définitive.

Et à partir d'un tableur (CALCSTAR, SUPERCALC, MULTIPLAN...) on peut éditer un tableau ou une partie de tableau en format texte et le reprendre dans un texte.

Ainsi, le traitement de texte n'est plus à considérer par le gestionnaire comme un accessoire, mais comme un tronc commun à partir duquel il pourra valoriser le résultat de ses traitements.

LE CAS WORDSTAR

Le logiciel WORDSTAR s'adapte à pratiquement toutes les situations:

- traitement de programmes
- lettres types et personnalisées
- mailings
- édition de rapports, d'ouvrages
- traitement complémentaire de données issues de SGBD ou tableurs

et toutes applications dont vous sentirez la nécessité. Toutes les commandes WORDSTAR sont accessibles à partir de codes de contrôle. La version WORDSTAR 2000 permet plusieurs milliers de combinaisons de commandes et dispose des extensions suivantes:

- mini tableur intégré.

- indexation par mot; on pointe un mot à indexer. A l'édition, WORDSTAR 2000 fabrique automatiquement un index.

- communication renforcée avec les autres programmes et les interfaces.

- mise en évidence des manipulations de bloc par coloration du texte à l'écran.

WORDSTAR 2000 est disponible sur les systèmes IBM XT et AT avec configuration mémoire de 256k minimum.

Pour édition laser ou matricielle, le logiciel de pointe pour l'édition de documents. Fonctionne sur IBM PC avec une configuration mémoire minimale de 512k, souris recommandée:

PERSONAL PUBLISHER fonctionne à l'aide d'icones à cliquer (l'esprit du Mac est passé par là). Les textes apparaissent tels qu'ils seront imprimés. Une fonction zoom permet de visualiser un détail du texte. Différentes fontes de textes sont disponibles et traitées dans des tailles variables.

L'insertion d'un dessin indente automatiquement le texte autour du dessin. Les dessins proviennent d'une bibliothèque préprogrammée ou définis par l'utilisateur.

PERSONNAL PUBLISHER ne traite pas les programmes.

LE CAS DU TO9

Encore une fois, l'esprit du Mac a frappé. Le TO9 ou TO9+ est équipé en standard d'un traitement de texte. Les textes sont affichés tels qu'ils seront imprimés. Le mixage texte-graphisme est possible. Les images devront être préalablement créées à partir du logiciel COLORPAINT. Ces images seront sauvegardées avec l'extension MAP.

La gestion des fonctions du traitement de texte intégré est réalisée à partir du clavier, de la souris ou du crayon optique.

On peut critiquer le fait que certaines options n'agissent pas localement. Ainsi, la justification à droite agit sur l'ensemble du texte lors d'un déplacement de marge. De même, la tabulation automatique en première ligne ne permet pas la composition de texte 'à l'américaine'.

CAS DU PCW 8256 ET PCW 8512

AMSTRAD a fait très fort en proposant un ensemble intégrant un traitement de texte très performant. On peut critiquer cependant l'obligation d'utiliser l'imprimante AMSTRAD, les fonctions de transcodage pour une autre imprimante étant inexistant.

AMSTRAD autorise le traitement de textes ASCII (listings source), mais les commandes ne se rapprochent d'aucun utilitaire connu. Le passage d'un traitement de texte AMSTRAD à un traitement de texte disponible sur une machine plus 'pro' sera peu évident.

→

LA DOCUMENTATION DES FICHIERS

De plus en plus souvent, la diffusion d'un logiciel du domaine public ou en freeware est accompagnée d'un texte joint au programme et disponible en tapant une simple commande:

```
TYPE fichier.ext
```

Ainsi, sur F83, le fichier de documentation est visible en tapant sous MSDOS:

```
TYPE README.PC
```

et sous CP/M:

```
TYPE FORTH.DOC
```

Ces deux fichiers ont été générés à partir de WORDSTAR (il y a des commandes 'point' dans les fichiers texte) et imprimés à l'aide de MAILMERGE sur le disque.

L'option d'impression sur disque permet ainsi de joindre une documentation complète à un produit sans recourir à la photocopie ou à l'imprimerie. A charge pour l'utilisateur d'imprimer par ses propres moyens le document en question (en utilisant WORDSTAR ou la commande TYPE sous CP/M ou MSDOS).

Une disquette de 360k remplace un document de près de 140 pages.

C'est également à l'aide d'un traitement de texte que sont générés les fichiers d'aide (extension HLP). Un sous-programme extrait les pages permettant d'aider l'utilisateur au niveau du programme considéré.

INTERACTION TRAITEMENT DE TEXTE ET TELEMATIQUE

Le MINITEL est un média dont la capacité d'information reste encore très peu exploitée. Il est tout à fait concevable de générer des rubriques reprenant le contenu de fichiers ASCII générés à partir d'un traitement de texte.

Pour ce faire, il faut à la saisie sélectionner sa marge de droite à 40 caractères et écrire son texte.

Plus tard, lorsque l'utilisateur désire consulter un bulletin, le système télématique ouvre le fichier contenant le bulletin en question et le diffuse par tranches de 20 à 24 lignes. Une pause en bas de page attend la frappe <suite> ou <retour> ou en option <page n>.

Ainsi, de nombreux textes peuvent être repris à défaut d'être saisis et être simplement formatés en 40 caractères par ligne. Dans le cas de transmission V21 (80 car/ligne) on pourra envoyer par paquet de deux lignes s'il n'y a pas de retour chariot dans le texte original.

A l'inverse, le texte de l'article proposé dans JEDI 32, pages 3 et suivantes, d'origine OUF a été enregistré au format ASCII depuis un modem avant d'être remis en forme à l'aide de WORDSTAR. Donc, si les listes de numéros sont erronnées, ce ne peut être qu'une erreur de saisie provenant de OUF et non de JEDI. Le programme de communication utilisé en la circonstance s'appelle MODEM7 (sous CP/M) et enregistre la transaction par activation de ^Y après avoir affecté un nom de fichier.

Ce procédé de réception de données a également permis de recevoir un fichier CP/M d'extension HEX, lequel a été débarrassé des présentations d'usage à l'aide de WORDSTAR.

Et pour finir, la liste des numéros de la page 3, JEDI 32, peut également être traitée et mise en forme à l'aide de WORDSTAR pour être 'digérée' par dBASE II. Exemple:

```
19 44 224 641 585   ABERDEEN   UK  V23   PRESTEL
```

devient après remise en forme:

```
19 44 224 641 585
ABERDEEN
UK
V23
PRESTEL
```

Le logiciel dBASE peut lire et traiter des fichiers provenant d'autres logiciels que dBASE. Dans ce cas précis, dBASE traite un fichier au format ASCII manipulé à partir de WORDSTAR.

Cette dernière manipulation est également possible avec toutes données ASCII provenant d'une transaction télématique:

- catalogues,
- cours de la bourse,
- annuaire téléphonique ou TELETEL (AST3 sur 3615). Je vous laisse imaginer ce que l'on peut faire en indexant ensuite un tel fichier et en y rajoutant ses propres données complémentaires.
- etc...

En BASIC, ces données peuvent devenir des DATAS. En effet, il suffit de remplacer à l'aide de WORDSTAR le retour chariot situé en début de ligne (fin de ligne précédente), ce qui donne dans le dernier exemple:

- appel de la fonction recherche/remplacement

```
SEARCH:^N                (taper ^P^N)
REPLACE WITH:^N DATA     (taper ^P^N DATA)
OPTIONS:5                (nombre de remplacements)
```

ce qui donne:

```
DATA  19 44 224 641 585
DATA  ABERDEEN
DATA  UK
DATA  V23
DATA  PRESTEL
```

Il ne reste qu'à mettre les numéros de ligne et à fusionner avec le reste du programme BASIC.

A noter que si vous vouler grouper plusieurs DATAS sur une même ligne, il suffit de taper:

- on pointe au début du premier DATA
- recherche/remplacement ^QA

```
SEARCH:^N  DATA       ( taper ^P^N  DATA  )
REPLACE WITH:,        ( taper ,           )
OPTIONS:4             ( à répéter 4 fois  )
```

```
DATA  19 44 224 641 585,ABERDEEN,UK,V23,PRESTEL
```

Cette opération peut être renouvelée autant de fois qu'il y a de groupes de DATAs.

Voilà. Maintenant, si à partir d'un listing BASIC contenant plein de DATAS que vous voulez récupérer sous dBASE, utilisez WORDSTAR:

```
DATA COCHON,VACHE,POUSSIN
```

et activez recherche/remplacement par ^QA

```
FIND:,
REPLACE WITH:^N
OPTIONS:2
```

ce qui donne:

```
COCHON
VACHE
POUSSIN
```

En conclusion, ce rapide survol a permis de démontrer qu'un traitement de texte est avant tout un utilitaire sérieux aux utilisations variées et non seulement un simple gadget.

Arthur C. CLARKE a écrit 2010 ODYSSEE II sur traitement de texte. Ainsi la boucle se referme, car on ne parle jamais aussi bien d'un ordinateur, surtout de HAL, qu'en les connaissant.

FORTH *Notation algébrique infixée*

par Marc PETREMANN

Dans le numéro 26, nous vous proposions un programme de gestion de variables locales dont le principal avantage permettait d'accroitre considérablement la lisibilité des programmes écrits en langage FORTH. Mais cette lisibilité ne serait pas complète si on ne permettait pas l'écriture des formules arithmétiques en notation algébrique conventionnelle.

Divers procédés ont été publiés, dont un proposé par C.H. MOORE (LE CONCEPT FORTH par P.COURTOIS en annexe, un compilateur BASIC simplifié). La méthode la plus simple publiée à ce jour est celle proposée par M.STOLOWITZ (FORTH DIMENSION) et que j'ai adapté au nouveau standard F83.

Les routines proposées ci-après fonctionnent aussi bien sur F83 CP/M que sur F83 MSDOS. Elles peuvent très facilement être adaptées aux standards FIG et 79-STANDARD. Le listing:

```
\ fonctions algebriques infixees
CREATE OP 44 ALLOT
: ?INTERP  ( pfa ---)   STATE @ IF  ,  ELSE  EXECUTE  THEN ;
: OPP@     ( --- adr)    OP DUP @ + ;
: >OP      ( pfa niveau ---)   4 OP +!   OPP@ 2! ;
: OP>      ( ---)     OPP@ 2@ -4 OP +!   DROP ?INTERP  ;
: LEV?     ( --- niveau)  OPP@ @  ;
: ]A   BEGIN LEV? WHILE OP> REPEAT FORTH ; IMMEDIATE

: TRAITE   ( pfa ---) 2@
BEGIN  DUP LEV? > NOT
WHILE  >R >R OP> R> R>   REPEAT  >OP ;

: INFIX    ( n --- <mot> <mot>   en compilation)
 ' CREATE SWAP , , IMMEDIATE
   DOES> TRAITE  ;


\ fonctions algebriques infixees
VOCABULARY ALGEBRE      ALGEBRE DEFINITIONS

7 INFIX * *     7 INFIX / /     6 INFIX + +     6 INFIX - -
5 INFIX > >     5 INFIX < <     5 INFIX = =
4 INFIX NOT NOT 3 INFIX AND AND 2 INFIX OR OR

: ( [']  CR 1 >OP ; IMMEDIATE

: )   [ FORTH ]  BEGIN  1 LEV? <  WHILE  OP>  REPEAT
 1 LEV? = IF -4 OP +!  ELSE TRUE ABORT" manque (" THEN ;
IMMEDIATE

FORTH DEFINITIONS

: A[   OP 44 0 FILL  ALGEBRE ; IMMEDIATE
```

Une fois le contenu de ces écrans compilé, vous pouvez écrire des formules algébriques en notation polonaise inverse, ce qui est la norme FORTH, ou en notation algébrique:

```
A[ ( 3 + 5 ) * ( 7 - 2 ) ]A .   affiche 40
```

ou

```
: TEST  ( — n)
A[ ( 3 + 5 ) * ( 7 - 2 ) ]A ;
```

Si on a la curiosité de décompiler la définition de TEST par SEE TEST, on obtiendra à l'affichage:

```
: TEST 3 5 + 7 2 - * ;
```

Les deux mots clés sont A[et]A et entre lesquels est écrite la formule en notation algébrique.

Attention, les niveaux de parenthèses sont limités par la capacité de l'espace mémoire réservé dans le tableau OP (écran 2, ligne 1). En cas de débordement, prévoir un espace plus grand qui soit un multiple de 4.

La notation algébrique ainsi définie ne perturbe aucun des mécanismes conventionnels de FORTH, que ce soit en interprétation ou en compilation. En outre, elle est applicable aux tests logiques:

```
...
A[ ( FLAG1 à  AND  FLAG2 à  ) OR TEST3 à ]A IF ...
```

En utilisant les mécanismes des variables locales, on peut écrire ce genre de test:

```
...
A[  TRUC < 5  AND  MACHIN = 0 ]A IF ...
```

Et l'affectation après une opération algébrique deviendrait:

```
...
A[ ACHATS * TVA / 100 ]A TO TAXES ...
```

Pour les barbus de la programmation, on pourra vectoriser les fonctions générées lors de la définition des opérateurs algébriques de manière à traiter des nombres simples précision, double précision ou en virgule flottante. Exemple:

si l'opérateur + est DEFERé, on l'initialise par

```
 ' + ALGEBRE IS + FORTH     en simple précision,
 ' D+ ALGEBRE IS + FORTH    en double précision,
 ' F+ ALGEBRE IS + FORTH    en virgule flottante.
```

Ainsi, l'initialisation des opérateurs peut être activée à partir des équivalents de A[et]A, que l'on peut écrire sous la forme suivante:

```
A[ et ]A    en simple précision,
D[ et ]D    en double précision,
F[ et ]F    en virgule flottante.
```

mots dont je vous laisse le soin de la définition. ■

Suite de la page 2

On se rend facilement compte, si l'on sauvegarde les valeurs sous forme litérale, que ce code ne pourra être recompilé qu'à l'adresse 8400h et suivantes. Il faudra analyser manuellement le code ainsi généré et pointer toutes les valeurs correspondant à des adresses susceptibles de varier. Ce travail effectué, la nouvelle version de CALC devient:

```
VARIABLE CALC -2 ALLOT HEX
34 C, 10 C, 37 C, 06 C, A6 C, 41 C,
8E C, ECRAN , 30 C, 86 C, 86 C,
SCRLAR C, 58 C, 58 C, 58 C, 3D C, 30 C,
8B C, AF C, C4 C, 35 C, 10 C, 39 C,
HERE CALC 2- !  BD C,  ' CALC ,
7E C, 0099 , DECIMAL
```

Or, ce travail, s'il reste rentable pour une ou deux définitions, ne peut être appliqué valablement à des programmes de plusieurs centaines d'octets. C'est pourquoi je reformulerai la question de M.ZUPAN en proposant l'étude d'un moyen d'implanter l'assembleur assez loin dans le haut de la mémoire, puis une fois les définitions en code machine assemblées dans le dictionnaire normal, de déconnecter l'assembleur pour récupérer la place prise en mémoire. Bon courage... ■

UN ASSEMBLEUR 6809 EN FORTH
par Marc PETREMANN

Diffusé depuis un an en disquette, l'ASSEMBLEUR FORTH 6809 pour THOMSON, adaptable aux autres systèmes équipés du 6809, n'avait qu'une documentation restreinte. Voici enfin ces explications détaillées dont la publication sera suivie du listing dans le prochain numéro de JEDI.

Créer un véritable assembleur autorisant la définition de macro-instructions, gérant des structures de contrôle, qui est intimement lié au vocabulaire d'un langage évolué, c'est à dire autorisant la mixité des instructions entre FORTH et l'assembleur, et capable de générer un code objet en une seule passe semble relever de l'utopie. C'est pourtant un tel assembleur que nous vous proposons dans ces pages. *suite page 15*

```
{$A+}
PROGRAM PRCMPRSS; {Affichage de fichiers texte compresses }
VAR
  BUFF : ARRAY [0..1151] OF BYTE;
  NOM1,NOM0  : STRING[16];
  CH,CH1     : BYTE;
  TAMPON     : ARRAY [1..128] OF BYTE;
  PTFICH     : BYTE;
  FICH       : FILE;
  PTBIT      : BYTE;
  SAVEBYTE   : BYTE;
  BYTECOUNT  : INTEGER;
  SAVEBIT    : BYTE;
  I          : INTEGER;
  LENTAB     : INTEGER;
  POSCUR     : BYTE;
  UNTILFLAG  : BOOLEAN;

 PROCEDURE OPENIN;
 BEGIN
   ASSIGN(FICH,NOM1);
   {$I-}
   RESET(FICH);
   {$I+}
   PTFICH:=129;
   PTBIT:=128;
   SAVEBYTE:=0;
   BYTECOUNT:=0;
   SAVEBIT:=0
 END;


 PROCEDURE READCH(VAR CH:BYTE);
 BEGIN
   IF PTFICH>128 THEN
   BEGIN
     BLOCKREAD(FICH,TAMPON,1);
     PTFICH:=1
   END;
   CH:=TAMPON[PTFICH];
   PTFICH:=SUCC(PTFICH)
 END;


 FUNCTION GETBIT : BOOLEAN;
 BEGIN
   IF PTBIT=128 THEN
   BEGIN
     READCH(SAVEBIT);
     PTBIT:=1
   END ELSE PTBIT:=PTBIT SHL 1;
   GETBIT:=(SAVEBIT AND PTBIT)<>0
 END;


 FUNCTION EVALUE : BYTE;
 LABEL FIN,LOOP;
 VAR   I     : INTEGER;
       FLAG  : BYTE;
 BEGIN
   I:=0;
 LOOP:
     IF GETBIT THEN I:=I+2;
     FLAG:=BUFF[SUCC(I)];
     IF FLAG>=128 THEN GOTO FIN;
     I:=BUFF[I] SHL 2;
  GOTO LOOP;
 FIN:
   IF FLAG=$FE THEN EVALUE:=$1A ELSE EVALUE:=BUFF[I] XOR -1
 END;
```

→

```
FUNCTION GETCHAR(VAR CH : BYTE) : BOOLEAN;
VAR
  CH1,CH2 : BYTE;
BEGIN
  IF BYTECOUNT=0 THEN
  BEGIN
    CH:=EVALUE;
    IF CH<>$90 THEN SAVEBYTE:=CH ELSE
    BEGIN
      CH1:=EVALUE;
      IF CH1=0 THEN SAVEBYTE:=CH ELSE
        IF CH1>1 THEN
        BEGIN
          BYTECOUNT:=CH1-2;
          CH:=SAVEBYTE
        END ELSE
        BEGIN
          CH:=EVALUE;
          SAVEBYTE:=CH
        END
    END
  END ELSE
  BEGIN
    CH:=SAVEBYTE;
    BYTECOUNT:=PRED(BYTECOUNT)
  END;
  IF CH=$1A THEN GETCHAR:=FALSE ELSE GETCHAR:=TRUE
END;


BEGIN
  IF PARAMCOUNT=0 THEN UNTILFLAG:=FALSE ELSE
  BEGIN
    UNTILFLAG:=TRUE;
    NOM1:=PARAMSTR(1)
  END;
  WRITELN;
  REPEAT
    IF NOT UNTILFLAG THEN
    REPEAT
      WRITE('*');
      READLN(NOM1)
    UNTIL NOM1<>'';
    OPENIN;
    IF IORESULT<>0 THEN
    BEGIN
      WRITELN('File not found');
      CLOSE(FICH)
    END ELSE
    BEGIN
      READCH(CH);
      READCH(CH1);
      IF (CH<>$76) OR (CH1<>$FF) THEN
      BEGIN
        WRITE(CHR(CH));
        WHILE CH1<>$1A DO
        BEGIN
          WRITE(CHR(CH1));
          IF (NOT EOF(FICH)) OR (PTFICH<129) THEN READCH(CH1) ELSE CH1:=$1A
        END
      END ELSE
      BEGIN
        READCH(CH);
        READCH(CH);
        NOM0:='';
        READCH(CH);
        WHILE CH<>0 DO
        BEGIN
          NOM0:=NOM0+CHR(CH);
          READCH(CH)
        END;
        WRITELN('Nom original : ',NOM0);
        READCH(CH);
        LENTAB:=CH;
        READCH(CH);
        LENTAB:=(LENTAB+CH*256)*4;
        IF LENTAB=0 THEN
```

→

```
        BEGIN
          WRITELN('ERREUR -- LENTAB=0!');
          HALT
        END;
        FOR I:=0 TO LENTAB-1 DO
        BEGIN
          READCH(CH);
          BUFF[I]:=CH
        END;
        WRITELN;
        POSCUR:=1;
        WHILE GETCHAR(CH) DO
        BEGIN
          IF CH=13 THEN BEGIN POSCUR:=1;WRITE(CHR(CH)) END
          ELSE
          BEGIN
            IF CH=9 THEN
            BEGIN
              I:=8-(POSCUR-1) MOD 8;
              FOR CH:=1 TO I DO WRITE(' ');
              POSCUR:=POSCUR+I
            END ELSE
            BEGIN
              WRITE(CHR(CH));
              IF CH<>10 THEN POSCUR:=SUCC(POSCUR)
            END;
            IF POSCUR>80 THEN POSCUR:=POSCUR-80
          END
        END
      END;
      CLOSE(FICH);
      WRITELN
    END
  UNTIL UNTILFLAG
END.
```

FORTH *Editeur pleine page (EPP)*

par B. LAMBEY

Il n'y a rien de plus éprouvant pour le malheureux qui n'a pas encore acquis son compatible que de voir s'amenuiser de jour en jour le nombre de pratiquants du bon vieux Forth-83, classique, éprouvé, mais éclipsé dans l'esprit de beaucoup par le F-83/Laxen-Perry pour IBM et consorts...
Quant à notre ami Petreman, il ne jure plus que par son compatible, semble-t'il,et il s'éloigne de tous les sans grades qui en sont encore à leurs vieux Z-80, 6809 et autres 6502, les paléographes!!

Pour tous ces antropopithèques dont je fais partie, voici, je l'espère la solution à un problème irritant, celui de l'édition pratique d'écrans Forth.

L'éditeur de ligne classique, est finalement assez lourd à manier. Un éditeur plein écran est tellement plus agréable que, lorsqu'on en a un, on éprouve guère le besoin de revenir à l'édition ligne à ligne. Voici donc un éditeur "Pleine Page" adapté aux habituelles 40 colonnes des machines courantes en France. Il peut être assorti d'utilitaires (Copie,TT).

Je suis parti du Forth-83 de Wilson M. FEDERICI, et de l'idée que sur 25 lignes de 40 colonnes, il y avait 1000 octets, au lieu des 1024 classiques (16 X 64).
Cela représente la perte des 24 derniers octets, mais sur un écran de Source, on en perd bien d'autres en "Blancs" d'agrémentation, pour l'indentation entr'autre. De plus sur une ligne où l'on case glorieusement un IF ou un BEGIN, on perdra moins de caractères si elle est de 40 colonnes au lieu des usuelles 64 colonnes. Au total on aura un écran plus lisible et un source plus "nourri" si on le souhaite ainsi, mais "qui peut le plus..."

Comme je voulais un soft aisément adaptable à d'autres machines, je n'y ai inclus aucune procédure de changement de couleurs, ou de suppression-restitution du curseur,qui rendent cependant le logiciel bien plus agréable d'emploi. Pour ceux qui désireraient une version couleur, j'en tiens une à leur disposition, celle que j'utilise avec mon Squale (qui a dit "Qu'est-ce que c'est que ça?").
Sur cette machine je dispose de 16 couleurs en direct et de 113 codes ASCII imprimables en plus des classiques 31 codes de commande de 0 à 31. J'ai donc utilisé comme clés d'accès aux ordres de l'éditeur les codes inhabituels, ou ceux qu'utilise le moniteur-machine, de cette manière on évite d'avoir une distorsion entre les commandes Editeur et le maniement courant du clavier. On trouvera dans le PG la liste des codes utilisés, dans les fonctions CASE/Type Taker qui ventilent les commandes sur le clavier.
Il doit être possible sans grande difficulté de remplacer les codes utilisés par d'autres, en cas de nécessité.

Il est, en tous cas, indispensable de rétablir dès l'implémentation un jeu d'instructions couleurs permettant, grâce à la couleur du curseur, de se rappeler quel est le mode d'action sélectionné. Dans l'état actuel, EPP indique son mode de travail par la position du curseur, mais c'est insuffisant, la confusion étant possible entre tous les modes d'écriture. A titre de suggestion je signale ci-après les couleurs que j'utilise sur la version Squale, sur "papier" vert olive:

```
Curseur marron = Lecture.
               = Ecriture normale.
Curseur rouge  = Mode "Insertion".
Curseur mauve  = Mode "Delete".
Curseur jaune  = Mode "Paquet" / entrée et sortie.
Curseur bleu   = Mode "Paquet" / sélection du paquet à transférer.
Curseur marine = Sortie de l'éditeur.
```

Listing →
Suite page 15

```
Ecran No   1 ( Compléments        1 /4 )

: VAR variable ;        : CTE constant ;
: DEC decimal  ;        : EXE execute  ;
: ¬      emit  ;        : BELL  7 emit ;
: CLS 12 emit  ;        : BS    8 emit ;
                        : RC   13 emit ;
  17 cte TIT            : LF   10 emit ;
  27 cte C-NAME         : 94¬  94 emit ;
  40 cte C/L            : 40+  c/l +   ;
1024 cte B/BUF          : 40-  c/l -   ;
 959 cte TOTEC

: \   ( -- )
  >in @ c-name - c/l / 1+ c/l *
  c-name + >in ! ; immediate  -->
\ --------------------------------------
\ Bloquera le compilateur jusqu'à la fin
\ de la ligne, permettant ainsi la mise
\ en place de commentaires non compilés,
\ juste en dessous du mot dont commence
\ la definition.
\ Les micro-définitions ci-dessus per-
\ mettent in gain de temps en cours
\ d'exécution du programme.
```

```
Ecran No   2 \ Compléments        2 /4

: HOME        ( -- )             \ Envoi du
  c/l dup 1 dup 6 5 0 do ¬     \ curseur
                       loop ; \ en haut à
                                \ gauche de
: AT          ( x y -- )        \ l'écran,
  home 1 3 0 do dec ¬ loop ; \ et place-
                                \ ment du
\ curseur depuis HOME. Ces 2 procédures
\ sont adaptées au moniteur du Squale.
\ --------------------------------------

: ><          ( n a b -- flg )
\ ---"True" si n inclus entre a et b----
      >r over r> < >r > r> and ;

: 2P-RT       ( nab - nnab )
\ Abréviation pour les compléments à la
\ structure CASE/Eaker.
\ --------------------------------------
     2 pick -rot ;

: NON         ( n1 n2 -- flg )
\ ------Inverse flag de l'égalité-------
     = 0= ; -->
```

```
Ecran No   3 \ Compléments        3 /4

: <OF       ( n -- )
\ Pour "n" plus petit  que la valeur qui
\ figure devant "<OF" ( Comme pour "<" )
\ --------------------------------------
  4 ?pairs compile over compile > \ Oui!
          compile ?branch >mark  \ ====
          compile drop 5 ; immediate

: >OF       ( n -- )
\ Pour "n" plus grand  que la valeur qui
\ figure devant ">OF" ( Comme pour ">" )
\ --------------------------------------
  4 ?pairs compile over compile < \ Oui!
          compile ?branch >mark  \ ====
          compile drop 5 ; immediate

: ><OF      ( n -- )
\ Pour"n"compris entre les 2 valeurs qui
\ figurent devant "><OF"  ( Voir: "><" )
\ --------------------------------------
  4 ?pairs compile 2p-rt compile ><
          compile ?branch >mark
          compile drop 5 ; immediate
-->
```

```
Ecran No   4 \ Compléments        4 /4

: Y/N         ( -- flg )
\ Laisse le flag -1 si "Oui", et sinon
\ ---laisse le flag 0. Est récursif-----
  cr 5 spaces
  ." [O]ui ou [N]on ?" space key
  dup dup ¬ 109 112 ><
      if 111 = cr
      else drop bell ."  ~o/n !" c/l 0
        do bs loop recurse
      then ;

 : ELD         ( -- )
\ -------Chargement écran courant-------
      scr @ load ;

var CPTR
: INCR  1 cptr +! ; : DECR -1 cptr +! ;
\ --------Ajustement du compteur--------

: ?FREE       ( -- )
\ --------Affiche mémoire libre---------
    s0 @ 256 - here - b/buf / cr
    ." Nombre de blocks libres : " .
    cr ; -->
```

```
Ecran No   5 \ Editeur PP-V3  1 /21

\            Variables utilisées.
\ --------------------------------------
var SCA               \ Pointeur écran
var CTR               \ 2ème Compteur
var NBS               \ Compteur
var SITE              \ Site du curseur .
var VEC               \ Vectorisation
var VEC1              \       de la
var VEC2              \      bascule
var VEC3              \      Read/Write.
\ --------------------------------------
 var EDPAD 24 c/l     \ Pad pour partie
          * allot    \ éditable écran.
                      \ ---------------
var LP    45 allot    \ Pad de ligne.
\ --------------------------------------

: RASEDPAD ( -- )
\ ---------Blanchit EDPAD---------------
    24 c/l * edpad over blank edpad ! ;

: CT@  cptr  @ ;      \ Abréviation.
: INSI site +! ; --> \ Actualise SITE
```

```
Ecran No   6 \ Editeur PP-V3  2 /21

: ECRAD ( -- adr ) scr @ block ;
: CURAD ( -- adr ) site @ ;
: DEBAD ( -- adr ) ecrad c-name + ;
: LIMAD ( -- adr ) debad totec + ;
: MEM   ( adr -- ) site ! ;
\ --------------------------------------
\ Jeu d' adresses-repères dans l'ordre :
\ écran, curseur, début, et fin d'écran.

: DEBU  ( -- ) debad      mem 1 02 at  ;

 : BOTM  ( -- ) limad 39 - mem 1 25 at ;
\ Positionnement du  curseur en début et
\ -------en fin travail éditeur.--------

: AL      ( n -- )
\ ----Détermine le nombre d'alinéas.----
     0 do cr loop ;

: NAME=  ( -- )
\ ------Accès à zone titre-écran--------
  14 1 at pad c-name expect
  pad  ecrad  c-name  cmove debu ;   -->
```

```
Ecran No   7 \ Editeur PP-V3  3 /21

: ADLICU      ( -- adr )
\ -------Adr l'oct. ligne curseur-------
      curad dup debad - c/l mod - ;
: ENDLI       ( -- adr )
\ -------Adr fin ligne curseur----------
      adlicu 40+ ;

: MEMLIN      ( -- )
\ -------Mémorise ligne dans LP---------
      adlicu dup Lp c/l
      cmove rc mem ;

: L>PAD       ( -- )
      Lp 42 blank memlin ;
: PAD>L       ( -- )
\ ----Entrée et sortie Pad de ligne-----
      rc Lp adlicu dup mem c/l cmove
      Lp c/l type bs rc  ;

: LASTLIN     ( -- flg )
\ Laisse "True" si curseur sort du cadre
\ --------------------------------------
      adlicu limad c/l - > ; -->
```

```
Ecran No   8 \ Editeur PP-V3  4 /21

: ENT    ( -- )
\ -------Affiche l'en-tête d'écran------
  home cls ." Ecran N°" scr @ 4 .r
  ecrad dup c-name + swap space
     do I c@ dup 31 148 >< not
          if drop bl I c! update
          else ¬
          then
     loop 1 2 at debad mem ;
: SORT?  ( n flg -- )
\ Actualise car. à la position curseur .
\ --------ou non selon le flag.---------
          if 2drop bell
          else insi ¬
          then ;
: <LIM   ( n -- )
     dup curad + debad < sort? ;
: >LIM   ( n -- )
     dup curad + limad > sort? ;  -->
\ --------------------------------------
\ Selon "n" , nombre de cases dont va se
\ déplacer le curseur,calculent s'il y a
\ sortie de l' écran considéré comme une
\ seule ligne de 960 caractères
```

```
Ecran No   9 \ Editeur PP-V3  5 /21

: DEB-FIN ( -- )
\ Bascule du curseur 1ère/dernière ligne
\ --------------------------------------
  debad totec 2/ + curad > if botm
                              else debu
                              then ;
: SORT   ( -- )
\ Quitte lect. ou écrit. (selon vecteur)
\ --------------------------------------
     botm exit ;
: ECRI   ( c@ -- c@ )
\ Mémorise le caractère entré au clavier
\ --------------------------------------
     dup dup 31 148 >< if curad c!
                          then ;
: BLINE  ( -- )
\ Blanchit ligne et mémor. ligne curseur
\ --------------------------------------
     rc c/l spaces adlicu dup mem
     c/l blank ;
: SUPLI  ( -- )
\ Supprime ligne,ramène curseur 1er car.
\ --------------------------------------
     bline bs rc ;  -->
```

```
Ecran No  10 \ Editeur PP-V3  6 /21

: MEMIN  ( -- )
\ Mémorise fin d' écran avec ligne curs.
\ ----------------------------------------
  rasedpad adlicu limad over - swap
  over edpad 2+ swap cmove edpad ! ;

: -LINE  ( -- )
\ Supprime ligne curseur, remonte écran.
\ ----------------------------------------
lastlin
  if supli
  else memin edpad dup c/l 2+ +
    adlicu dup mem 2 pick @ 40- cmove
    rc dup c/l 2+ + swap @ 40- type
    limad c/l 1- - c/l blank c/l spaces
    1 curad debad - c/l / 2+ at
  then ;

: MEMEX  ( -- )
\ -----Décale contenu EDPAD de C/L.-----
  edpad dup dup 2+ adlicu dup
  mem 40+ rot @ 40- cmove
  dup 2+ swap @ 40- type
  1 curad debad - c/l / 2+ at ;   -->
```

```
Ecran No  11 \ Editeur PP-V3  7 /21

: VEXEC  ( n -- ) scr +! vec @ exe ;
\ Incrémente "SCR" pour exécuter vecteur
\ ----------------------------------------
: +LINE  ( -- )
\ Blanchit ligne curseur et décale écran
\ -----------vers le bas.---------------
          lastlin if supli
                  else memin bline memex
                  then     ;
: KOACUR ( -- )   curad c@ , bs ;
\ Emet le caractère  pointé par le curs.
\ ----------------------------------------
: RZ>LI  ( -- )
\ Efface fin de ligne, après le curseur.
\ ----------------------------------------
  0 endli curad do bl dup I c! , 1+
              loop 0 do bs
                     loop ;
: INCVAR ( -- )
\ Si possible incrémente SITE du curseur
\ ----------------------------------------
  curad 1+ limad > if bell
                  else 1 insi
                  then ;        -->
```

```
Ecran No  12 \ Editeur PP-V3  8 /21

: UCUR   ( n -- ) -40 <lim ;
: DCUR   ( n -- ) c/l >lim ;
: LCUR   ( n -- )  -1 <lim ;
: RCUR   ( n -- )   1 >lim ;
\ Mouvements curseur vers haut (up), bas
\ (down), gauche (left), droite (right).
\ ----------------------------------------
: VALID  ( n -- )
\ ---Prépare validation après travail---
               drop update ;
: ALINEA ( n -- )
\ Passe à la ligne si c'est possible   .
\ ----------------------------------------
  adlicu 40+ dup limad > if drop bell
                         else mem , if
                         then ;
: TOBOT  ( -- )
\ Placement curseur à la sortie éditeur.
\ ----------------------------------------
  save-buffers 1 24 at ;

: REDEX  ( -- )
\ -------Exécution vectorisée-----------
         vec1 @ vec ! vec3 @ exe ; -->
```

```
Ecran No  13 \ Editeur PP-V3  9 /21

: LI>PA  ( -- ) curad Lp ct@ cmove ;
: PA>LI  ( -- ) Lp curad ct@ cmove ;
\ Aller ou retour ligne (-> pad de ligne
\ des caractères qui suivent le curseur.
\ ----------------------------------------

: ?DEB   ( -- )
\ -------Stoppe en fin d'écran.---------
  curad 1+ limad = if debu     exit
                   then ;
: ?=BL   ( -- flg)
\ ----"True" si curseur sur un blanc----
        curad c@ bl = ;
: COMPTE ( -- )
\ Compte les car. jusqu'en fin de ligne.
\ ----------------------------------------
  0 cptr ! endli curad
      begin over over = not
      while incr 1+
      repeat 2drop ;
: RECPAD ( ) Lp 45 blank compte ;
: ACTUAL ( ) recpad li>pa pa>li ;
\  Blanchissage et actualisation du pad.
--> \-----------------------------------
```

```
Ecran No  14 \ Editeur PP-V3  10/21

: TO>SP  ( -- )
\ Saute au prochain blanc suivant un mot
\ ----------------------------------------
      begin ?deb ?=bl
      while 9 rcur recpad
      repeat begin ?deb ?=bl not
             while 9 rcur recpad
             repeat ;

: DELCAR ( -- )
\ Efface un caractère, et  entraine vers
\ la gauche le reste de la ligne.
\ ----------------------------------------
 curad adlicu >
 if koacur recpad li>pa
  begin bs -1 insi pa>li Lp ct@ type
    space bl ct@ curad + c! ct@ 1+ 0
       do bs
       loop
    key 127 non
    adlicu curad = or
  until
 then koacur ;   -->
```

```
Ecran No  15 \ Editeur PP-V3  11/21

: RAZEND  ( -- )
\ "Blanchit" la fin de l'écran depuis la
\ ligne où se trouve le curseur,celle-ci
\ étant incluse. La zone effacée est co-
\ piée dans EDPAD, à disposition.
\ ---(15 , efface la ligne courante)----
  memin lastlin
     if supli
     else rz>li limad
        adlicu c/l +
          do bl I c!
          loop 15 ,
     then ;

: KOAPAD  ( -- )
\ Affiche le contenu de "EDPAD" au début
\ de ligne où se trouve le curseur.
\ ----------------------------------------
  edpad 2+ >r adlicu limad over -
  rc  dup r@ swap type
  r> -rot cmove debu ;  -->
```

```
Ecran No  16 \ Editeur PP-V3  12/21

: INSCAR  ( -- )
\ ----------Mode insertion.-------------
  koacur recpad li>pa
    begin key dup 254 = not
    while dup 31 148 ><
      if curad dup adlicu 1-
         dup c/l + ><
        if drop ecri , incvar decr pa>li
           Lp ct@ type ct@ 0 do bs
                            loop
        else 2drop
        then
      else dup
           case 8 of lcur   actual endof
                 9 of rcur   actual endof
                10 of dcur   actual endof
                11 of ucur   actual endof
                13 of alinea actual endof
           endcase
      then
    repeat drop koacur ;         -->
```

```
Ecran No  17 \ Editeur PP-V3  13/21

var PDEB     \ Pointeur sur début paquet

: <PAQ    ( -- )
\ Repérage début du paquet à translater.
\ ----------------------------------------
      koacur adlicu pdeb ! ;

: PAQ> ( -- )
\ Repérage de fin du paqu   à translater
  ----------------------------------------
  adlicu c/l 1- + pdeb @ swap 2dup <
      if over - edpad 2+   2dup >r >r
         swap cmove r> r> 2- ! koacur
      else 2drop 1 2 at 0 pdeb !
           debad mem koacur
      then ;

: PAQ^    ( -- )
\ Restitution du paquet à nouvelle place
\ ----------------------------------------
  edpad 2+ dup adlicu over 2- @
  dup >r cmove r@ type r> insi
  rc adlicu mem ;   -->
```

```
Ecran No  18 \ Editeur PP-V3  14/21

: PAQUET  ( n -- )
\ Translation  d'un paquet  de "Source".
\ formé d' un groupe lignes repérées par
\ " Ctrl+( " pour 1ère ligne et "Ctrl+}"
\ pour la ligne de fin.On ressort le pa-
\ quet à la  nouvelle  place curseur par
\ "Ctrl+^" .    sage possible aux écrans
   ..ediatement précédent ou suivant.
\ ----------------------------------------
  koacur rc adlicu mem
      begin key dup
        case
  ( cur^ )  10 of dcur endof
  ( curv )  11 of ucur endof
  ( ct+( )  28 of valid -1 vexec endof
  ( ct+) )  30 of valid  1 vexec endof
  ( ct+^ )  94 of paq^ drop endof
  ( ct+{ ) 123 of <paq drop endof
  ( ct+} ) 125 of paq> drop endof
  ( ct+ES) 254 of drop koacur
                  exit endof
           drop \ Dans les autres cas---
        endcase
      again ;                   -->
```

```
Ecran No  19 \ Editeur PP-V3  15/21

: NOMME:     ( 1er-écran Last-écran -- )
\ Baptise n écrans du même titre et leur
\ -----attribue un No consécutif--------
  Lp c/l 2/ blank swap dup sca !
  over swap - 1+ nbs !
  Lp tit expect  1+ sca @
     do I block dup
        c-name blank
        124 over c!
        bl over 1+ c!
        Lp over 2+ dup >r
        span @ cmove
        bl r> span @ + dup ctr !
        1+ c! I 1+ sca @ - dup 99 >
           if ." Nb screen trop grand!"
              cr 2drop leave
           else 0 <# #s #> ctr @ 2+
              swap cmove
              47 ctr @ 4 + c! nbs @
                0 <# #s #> ctr @ 5 +
              swap cmove drop
           then update save-buffers
     loop ; -->
```

```
Ecran No  20 \ Editeur PP-V3  16/21

: COM ( n ) case ( cu~) 8 of lcur endof
  ( cu>)   9 of rcur endof
  ( cuv)  10 of dcur endof
  ( cu^)  11 of ucur endof
  ( RET)  13 of alinea endof
  ( ESC)  27 of valid tobot quit endof
  ( DEL) 127 of delcar drop endof
  ( ^N )  14 of koapad drop endof
  ( ^O )  15 of razend drop endof
  ( ^P )  16 of supli drop endof
  ( ^Q )  17 of pad>l drop endof
  ( ^S )  19 of -line drop endof
  ( ^T )  20 of rz>li drop endof
  ( ^Y )  25 of memlin drop endof
  ( ^Z )  26 of +line drop endof
  ( ^( )  28 of valid -1 vexec endof
  ( ^->)  29 of redex drop endof
  ( ^) )  30 of valid  1 vexec endof
  ( ^? )  31 of deb-fin drop endof
  ( ^' )  96 of drop paquet  endof
 (^S4 ) 130 of to>sp drop endof
 (^S0 ) 137 of name= drop endof
  ( ^Es) 254 of inscar drop endof
 ecri ¬ incvar endcase ;        -->
```

```
Ecran No  21 \ Editeur PP-V3  17/21

: ~WRIT ( -- )
\ Procédures d'écriture sous Editeur PP
\ cf: COMMANDES ci-devant.
\ --------------------------------------
  1 2 at debad mem
     begin key dup com
     again ;

: ~READ ( -- )
\ Procédures de lecture sous Editeur PP
\ commandes notées ci-dessous.
\ --------------------------------------
  c/l 25 at
          begin key
  ( Esc )    case 27 of tobot quit endof
  ( cur~)          8 of    -1 vexec endof
  ( cur>)          9 of     1 vexec endof
  ( ^R  )         23 of vec2 @ vec !
                               ~writ endof
           endcase
          again ;        -->
```

```
Ecran No  22 \ Editeur PP-V3  18/21

: AFECR  ( -- )
\ Affiche l'écran pour lecture/écriture.
\ --------------------------------------
  ent debad dup mem dup totec + swap
      do I c@ ¬ loop ;

: READ   ( -- ) afecr ~read ;

: WRITE  ( -- ) afecr ~writ ;

: RE     ( n -- ) scr !
\ Vectorisation passage écriture>lecture
\ --------------------------------------
        ['] read vec  ! 0 vexec ;

: WW     ( n -- ) scr !
\ Vectorisation passage lecture>écriture
\ --------------------------------------
        ['] write vec  ! 0 vexec ;

: 3BUFS  ( -- )
\ Passage  de 2 à 3  buffers permanents.
\ --------------------------------------
     3 ['] #BUFS >BODY ! ; -->
```

```
Ecran No  23 \ Editeur PP-V3  19/21

: ERE  ( -- ) scr @ re ; \ Pour lire
                         \    et
: EWR  ( -- ) scr @ ww ; \ pour écrire
\ ====================================
\ Ces deux commandes incluses on dispose
\ d' un jeu de quatre commandes pour en-
\ trer et sortir de l'Editeur,trois mots
\ et un touche :
\ ------------------------------------
\ =>       POUR ENTRER EN EDITION     <=
\ ------------------------------------
\   ERE = écran courant,lecture seule.
\   EWR = écran courant en écriture.
\ ------------------------------------
\ =>       POUR SORTIR DE L'EDITION   <=
\ ------------------------------------
\   Esc = Quitte édition,retour à Forth.
\   ELD = Sous Forth, charge l'écran
\         courant.
\ ====================================

  ' READ   VEC1 ! ' WRITE VEC2 !
  ' ~READ  VEC3 !   3BUFS   -->
\ Chargement initial des vecteurs.
```

```
Ecran No  24 \ Editeur PP-V3  20/21

312 cte BLK/DRV

: DR0    0    offset ! ;
: DR1 blk/drv offset ! ;

: (DO-BL)     ( ecr/fin ecr/deb -- )
\ -------"Blanchissage" d'écrans--------
  dup 0= if drop 1
         then
     do cr ." Blanchit " I 3 .r
        I block b/buf blank
        update save-buffers
     loop ;

: BLDSK       ( -- )
\ Blanchira tous les écrans d'un disque,
\ placé en DR1, en s'adaptant au format.
\ --------------------------------------
  cls ." Attention Système en DR0? OK?"
  y/n if dr1 blk/drv 1 (do-bl)
     then ;
-->
```

```
Ecran No  25 \ Editeur PP-V3  21/21

: BLSCR      ( ecr ecr2 -- )
\ --"Blanchit" depuis "ecr" à "ecr2"----
   dr1 1+ swap (do-bl) ;

: WIPE       ( -- )
\ -----Blanchit l'écran courant.-------
  dr1 scr @ dup blscr ;

: COPIES ( orig fin cible -- )
\ Copie x écrans de "orig" à "fin" vers
\ -----"cible" jusqu'à "cible+x"--------
  -rot 1+ swap
    do I over 2dup cr ." Copie "
             swap 3 .r ."  sur " 3 .r
       copy update save-buffers 1+
    loop flush drop ;
```

```
Ecran No  26 \ Commandes Edition  1 /13

======================================
1) LECTURE
 Le curseur  "Edition/Lecture" est en bas
et à droite de l'écran.
Seules répondront (Sécurité) les touches
suivantes :

"Escape"  Fin de lecture,retour à Forth.
--------  Le curseur passe en bas/gauche

 POSSIBILITES de MODIFS :

"Curseur vers gauche": ->Ecran précédent
 ------------------
"Curseur vers droite": ->Ecran suivant .
 ------------------
 CTRL+W : Passage de lecture à écriture.
 -------- le curseur passe en haut et à
          gauche. On peut commencer à
          éditer l'écran courant.

Toutes les autres touches seront invali-
dées ce qui évite toute manoeuvre intem-
pestive.
```

```
Ecran No  27 \ Commandes Edition  2 /13

======================================
2) ECRITURE
 Le curseur "Edition/Ecriture" est situé
en haut à gauche. Commandes de l'édition
par ls touches suivantes :

"Curseur vers gauche":       dito
"Curseur vers droite":       dito
"Curseur vers haut  ":       dito
"Curseur vers bas   ":       dito

-Commandes Simples   :
--------------------
"Escape"  Fin d'écriture,retour à Forth.
"Return"  Passage à la ligne.
"Delete"  Passage en mode "Delete"

-Commandes à deux touches:
-------------------------
Toutes  les  touches  désignées ci-après
sont frappées avec appui simultané sur
la touche Contrôle. Cela est rappelé
par l'accent circonflèxe conventionnel.
(Emploi de Majuscules ou Minuscules.)
```

Ecran No 28 \ Commandes Edition 3 /13

^N Restitue la fin d' écran mémorisée par le suivant , et ce à partir de la place actuelle du curseur , sur l'écran choisi (le même ou autre). Si le curseur est plus bas qu'à la position de mémorisation , le trop plein de lignes n'est pas affiché, mais le contenu du tampon spécial n'étant pas modifié , elles y sont toujours accessibles par le renouvellement de la commande.

^O Efface toute la fin d'écran depuis le début de ligne où se trouve le curseur . Mais tout ce qui est effacé est mémorisé et disponible pour un rappel par ^N . On mémorise pour recopier ailleurs par la succession ^O et ^N. (Sans toucher au curseur qui s' est placé en début de ligne automatiquement).

Ecran No 29 \ Commandes Edition 4 /13

^P Blanchit la ligne où se trouve le curseur,qui est envoyé en début de ligne, sans en sauver le contenu. Ne touche pas au reste de l'écran.

^Q Restitue la ligne mémorisée par ^Y en écrasant l'actuel contenu de la ligne où se trouve le curseur.Renvoie celui-ci en début de ligne.

^S Supprime la ligne où se trouve le curseur,qui disparait. Fait remonter d'une ligne toute la fin de l' écran, et le curseur réaparait en début de la ligne qui a remplacé celle qui a été effacée.

^T Le curseur reste en place et toute la fin de ligne est effacée , sans sauvegarde.

Ecran No 30 \ Commandes Edition 5 /13

^Y Mémorise le contenu de la ligne où se trouve le curseur,qui se trouve renvoyé au début, la ligne n'étant pas modifiée.

^Z Le curseur disparait. A son ancien emplacement est crée un ligne de blancs, tout l'écran est décalé d'une ligne vers le bas, y compris la ligne où était le curseur, et celui-ci réapparait en début de la ligne blanche créée. Si la dernière ligne (l= 24) était occupée, elle est perdue.

^> Passage à l'écran suivant en restant en écriture.Sauve le travail.

^< Passage à l'écran précédent en restant en écriture.Sauve aussi. Dans les deux cas le curseur se positionne an haut à gauche ce qui est la marque de "Ecriture".

Ecran No 31 \ Commandes Edition 6 /13

^Cu> (Soit Ctrl+"Curseur à droite) : Quitte le mode "Ecriture" et passe en mode "Lecture". Attention: SANS SAUVEGARDE de l'écran qui vient d'être édité ce qui permet de revenir au contenu ancien de l'écran après avoir sauvé le travail qui vient d'être fait (^O avant ^Cu> puis Esc , EE pour vider les buffers, ERE pour revenir en lecture, choix d'un écran vide, ^W pour revenir en "Ecriture", ^N pour recopier le travail récent , et retour à l'écran dont on aura voulu garder l'ancienne version.) .

^? Bascule 1ère-case-1ère-ligne . 1ère-case-dernière-ligne .

^' Passage en Mode "Paquet" .

^Esc Passage en mode "Insertion" . Cf:Mode "Paquet"et mode"Insertion"

Ecran No 32 \ Commandes Edition 7 /13

Commandes à trois touches:

Ces commandes nécessitent l'appui simultané de Ctrl+Shift (ci-après: C+S) et d' une troisième touche:

C+S+4 Saute-mouton,depuis la place actuelle du curseur, de mot en mot. Il ne fonctionne que de gauche>droite Pour reprendre au début de ligne , faire "Return" et taper un coup de curseur vers le haut.

C+S+# Passage en mode "TITRE". cf:mode "titre".

Pour toutes ces commandes : codes ASCII disponibles (décimal) sur écran 20/Ecr. et 21/Lecture.

Ecran No 33 \ Commandes Edition 8 /13

Commandes ECRITURE
MODE INSERTION.

C'est une Bascule.
Pour y accéder et en sortir: Ctrl+Escape
La frappe des touches positionnant le curseur, et l'alinéa par "Return", sont les seules qui ne donnent pas lieu à affichage. Toutes les autres touches sont normalement actives,mais en mode Insertion, et les lettres qui disparaissent en fin de ligne à droite sont définitivement perdues.

La couleur du curseur devient rouge comtout caractère inséré. C'est la marque du mode "Insertion". A toute relecture ultérieure, les caractères auront repris leur teinte normale: marron sur papier vert-olive clair.

Ecran No 34 \ Commandes Edition 9 /13

Commandes ECRITURE
MODE TITRE

On y entre par la commande C+S+# .
On en sort par "Return".

C'est un EXPECT de 27 caractères.Le curseur n'est donc pas "transparent" et les seuls caractères mémorisés seront ceux que vous allez taper ; Seront invalidés tous les caractères qui étaient en place (ancien titre) et que vous auriez pu avoir l'impression de "laisser".

Tout titre doit commencer par le caractère dit "Scratch" c-à-d "\" qui bloquera la compilation depuis le scratch inclus jusqu' à la fin de la ligne-titre.

Voyez aussi le titrage automatique des séries d'écrans consécutifs par la fonction NOMME: accessible hors "Ecriture".

Ecran No 35 \ Commandes Edition 10/13

Commandes ECRITURE
MODE PAQUET

On entre par la commande " Ctr+' "

Le mode Paquet est prévu pour sélectionner un paquet de source qu'on va définir afin de le translater ailleurs.

Huit commandes,toutes autres invalidées:

Cur^ Montée d'une ligne.
CurV Descente d'une ligne.
Ctr+Esc sort du mode Paquet > Ecriture.
Ctr+(Sélection début paquet
Ctr+) Sélection fin du paquet
Ctr+^ Le paquet est restitué à la nouvelle place du curseur.Ecrasement de ce qui était à la place que va occuper le paquet . (Le prévoir!)
Ctr+> Passage à l'écran suivant.
Ctr+< Passage à l'écran précédent.

Ecran No 36 \ Commandes Edition 11/13

Commandes ECRITURE
MODE DELETE

On y entre par la touche "Del".
On en sort en tapant toute autre touche.

En mode "Delete", toute frappe sur la touche "Del" fait reculer d' une case la série de caractères comprise entre fin de ligne et curseur.
Le début de ligne constitue une butée que le curseur-locomotive ne peut franchir.
A cet endroit si l' on insiste sur la touche "Del" cela a le même effet que la frappe d'une autre touche,fait sortir du mode Delete.
On est revenu en "Ecriture normale" .

Ecran No 37 \ Commandes Edition 12/13

Pour passer de Forth à l' Editeur pleine page on dispose de quatre mots :

ERE Passe sous "Editeur PP" en Lecture seule . Affichage de l' écran courant . (Celui dont le No est dans la variable système SCR).

EWR Passe sous Editeur PP en Ecriture avec marron en haut à gauche. C'est l'écran courant qui est offert à l'édition.

RE (n --) Passe en LECTURE de l' écran dont le numéro a été empilé.

WW (n --) Passe en ECRITURE de l'écran dont le numéro a été empilé.

NOTE Lorsqu'on vient de quitter Editeur PP par " Escape " on peut charger l'écran courant par ELD .

Ecran No 38 \ Commandes Edition 13/13

Cet éditeur "PP" offre un certain nombre d'utilitaires,qui sont expliqués dans le Source. Il s'agit de :

NOMME: (1er-écr dern-écr --) qui baptise les n écrans séparant les deux numéros empilés , ceux-ci inclus , d' un même nom . Ajoutera au nom un numéro d'ordre et, après une barre de fraction, le total de la série baptisée. Il est impératif de taper "Return" après le ":" de NOMME: ,et un nouveau "Return" dès la fin du titre.

Sont aussi disponibles:

WIPE (--) COPY (a b --)
BLSCR (ec1 ec2 --) COPIES (a b n --)

Suite de la page 10

De plus la possibilité de supprimer le curseur avant toutes les tâches d'affichage augmente beaucoup lisibilité et rapidité des dits affichages. Bien entendu on rétablira le curseur à la fin de chaque tâche. Cela aussi doit être installé.

J'ajoute que les compléments à la structure "Eaker's CASE" simplifient l'emploi de celle-ci.

Y/N, quant à lui, est récursif et ne décale pas l'affichage, en cas d'erreur de frappe à la réponse.

Je tiens enfin à rendre hommage au remarquable travail du "Dear Wilson" qui est très satisfait de savoir que son 6809/Forth-83/Flex est connu, utilisé et apprécié en France. Il est de bon conseil, et ne manque jamais de répondre aux courriers, surtout quand ils demandent un conseil. Il vient de mettre au point un utilitaire de transfert de groupes d'écrans sur le disk, qui utilise comme buffer le Dictionnaire! Il n'est pas piqué des hannetons! je le communiquerai à ceux que ça intéresse.
De même,mon traitement de texte, élaboré pour une MT-80, travaillant en francais, et adaptable simplement à toute autre imprimante est aussi à la disposition de ceux qui en feront la demande à JEDI. Il est en Forth, ai-je besoin de le préciser!

Enfin, ma modestie ne souffrira pas de toute critique ou suggestion, car je sais très bien que EPP est améliorable. Et je serai heureux de savoir ce que les uns ou les autres proposent.
Amicalement à tous.

Bernard LAMBEY

Adresse utile pour les adeptes du 6809:

Wilson M. FEDERICI
1208 NW Grant
Corvallis OR 97330
USA

Avec notre assembleur écrit en FORTH, vous pourrez créer des mots dont la définition peut faire moins de dix octets et vérifier leur bon fonctionnement de suite, puis les intégrer en temps que sous-programmes dans des définitions plus générales. Attention, cet assembleur n'inclut pas de moniteur, mais le langage FORTH dispose d'aides à la mise au point qui ne sont pas inintéressantes.

Suite de la page 8

CONCEPT DE BASE

L'idée de départ est de substituer du code machine à toute valeur représentant un opérateur, en exécutant un mot chargé d'implanter ce code dans le dictionnaire. Exemple:

```
HEX
: rts  39 C, ; DECIMAL
```

Mais définir sous cette forme tous les mnémoniques d'assemblage risque d'être une opération longue et encombrante. Il est plus pratique de définir un mot de création de mnémoniques:

```
: INH
CREATE C,
 DOES> C@ C, ;
HEX 39 INH rts DECIMAL
```

Cette apparente simplicité de définition, applicable aux opérateurs utilisant l'adressage inhérent, n'est pas aussi aisément applicable aux autres modes d'adressage. Il faut considérer trois catégories principales d'opérateurs en fonction de leurs modes d'adressages:

- ceux ne nécessitant aucun opérande. C'est le cas des opérateurs utilisant l'adressage inhérent.
- ceux qui sont utilisés en adressage direct, étendu ou indexé exclusivement.
- ceux qui sont utilisés en adressage immédiat ou en commun avec les adressages direct, étendus ou indexé.

Ne sont pas inclus les opérateurs de branchement conditionnel et inconditionnel.

Si la définition des opérateurs travaillant en adressage immédiat, étendu et direct ne pose guère de problème, il n'en est pas de même pour l'adressage indexé. La résolution de toutes les combinaisons de ce mode d'adressage, très riche en possibilités, représente une bonne part du programme assembleur.

suite page 17

LE CHOIX D'UNE SYNTAXE ASSEMBLEUR

Cet assembleur n'utilise pas une syntaxe conventionnelle. Au premier abord, le choix d'une syntaxe spécifique à un

FORTH

LA MAITRISE DU GRAPHISME
par Marc PETREMANN

Les systèmes THOMSON, comme la plupart des micro-ordinateurs actuels, prennent en considération deux modes d'affichage, le mode texte et le mode graphique. Sur certains systèmes, tel APPLE II ou ORIC, ces modes sont distincts. D'autres systèmes, et y copris les systèmes THOMSON, affichent simultanément les informations en mode texte et en ode graphique. On peut mélanger les textes et les dessins. Le nombre d'éléments affichables en mode texte dépend du rapport entre la taille d'un élément en mode texte et la taille de la zone d'affichage disponible sur l'écran vidéo.

Si on regarde de très près un caratère quelconque affiché à l'écran (essayez /), on constate qu'il est constitué d'une série de points ordonnés. Pour les systèmes THOMSON TO7, TO7-70 et MO5, les caratères affichables sont codés dans une matrice carrée contenant 64 points.

L'emploi des seuls caratères alphanumériques ne permet pas d'apporter une plus value à la présentation des informations affichées à l'écran. Ne dit-on pas qu'un petit dessin peut remplacer un long discours. C'est pourquoi on a recours aux fonctions graphiques.

Sur les systèmes THOMSON sous FORTH, l'allumage d'un point est obtenu en utilisant le mot PSET (x y PSET) avec x compris dans l'intervalle 0..319 et y dans l'intervalle 0..199. Exemple:

```
30 50 PSET
```

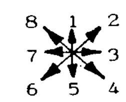
Cette notion de pixel (ou élément graphique) est importante, car elle recoupe cellme de bit en logique binaire. La maitrise des mécanismes discrets qui contrôlent l'affichage permet d'obtenir des effets impossibles à réaliser par les fonctions pré-programmées du système.

Il y a quelque temps, j'ai demandé dans la lettre du secrétaire des informations concernant l'implantation des caratères arabes et cyrilliques sur les claviers de machines à écrire. Je tiens ici à remercier l'adhérent qui a répondu à ma demande. Ma première idée était d'élaborer un programme de traitement de texte en arabe. Puis, par manque de temps, je n'ai pu qu'ébaucher une première idée des mécanismes particuliers à ce programme dont voici les principes.

Pour définir un alphabet graphiquement différent de celui disponible dans le système, plusieurs solutions s'offrent à nous. La première consiste à créer tous les caractères dans des matrices binaires. Mais cette solution à ses limites: nombre de codes disponibles limité, taille de la matrice réduite. Essayez un caratère KANJI (japonais) dans un carré 8x8...

La solution qui devait être retenue est celle du codage de FREEMAN. Cette méthode permet de définir un dessin à partir d'un point et une série de codes représentants les mouvements élémentaires. A l'exécution, ce codage s'apparente au dessin à main levée.

Définition des codes de mouvement:



Puis on choisit un dessin à représenter et on reporte sa représentation à une échelle suffisament grande dans une grille Prendre un point de départ P et avancer case par case en notant le code du déplacement.

Pour exemple, j'ai choisi la lettre N majuscule manuscrite et implanté les codes dans la zone paraètre du mot LETTRE-N. Cette séquence de codes est terminée par le code 0 qui signifiera à l'exécution la fin de liste. ➡

PROLOG

```
/*********************************************************/
/*      PROGRAMME DE GENERATION DE GRILLES DE LOTO       */
/*ECRIT EN TURBO-PROLOG PAR JEAN-PAUL POSTEC   AVRIL 1987 */
/*********************************************************/

domains
  liste=integer*

predicates
  hasard(integer)
  tirage(liste)
  différent(liste)
  hors(integer, liste)
  répète(integer)
  tri(liste, liste)
  ajout(liste, liste, liste)
  écrireliste(liste)

/*****************************************/
/* GENERATION DE 8 GRILLES D'OU REPETE(8) */
/*****************************************/

goal
  répète(8).

clauses
  hasard(X) if random(Y), X=1+48*Y.

/* Tantque les 6 nombre tirés au hasard ne sont pas
   différents on recommence le tirage */

  tirage([A,B,C,D,E,F]) if hasard(A), hasard(B),
                           hasard(C), hasard(D),
                           hasard(E), hasard(F),
                           différent([A,B,C,D,E,F]),!.
  tirage([A,B,C,D,E,F]) if tirage([A,B,C,D,E,F]).

/* Une liste a tous ses éléments différents si d'une part
   elle est vide et d'autre part si sa tête n'est pas
   élément de sa queue et si tous les éléments de sa queue
   sont eux-mêmes différents */

  différent([]) if !.
  différent([X|Y]) if hors(X,Y), différent(Y).

/* N'importe quoi est hors d'une liste vide, de plus
   X n'est pas élément d'une liste s'il est différent
   de la tête de liste et s'il n'est pas élément de
   la queue de liste  */

  hors(_,[]) if !.
  hors(X,[Y|Z]) if X<>Y, hors(X,Z).

  répète(0) if !.
  répète(N) if tirage([A,B,C,D,E,F]),
               tri([A,B,C,D,E,F],Liste_triée),
               écrireliste(Liste_triée),
               nl, N1=N-1, répète(N1).

/* Il s'agit ici du tri bulle utilisant la concaté-
   nation (ajout) de deux listes */

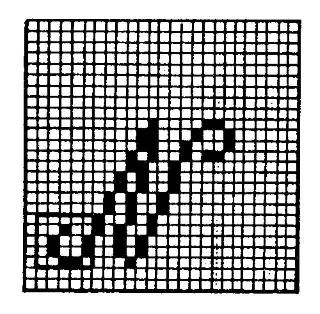
  tri(L1,L2) if ajout(X,[A,B|Y],L1),
                B<A, ajout(X,[B,A|Y],L3),
                tri(L3,L2), !.
  tri(L,L).

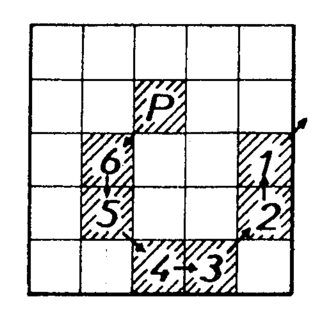
  ajout([],L,L).
  ajout([X|L1],L2,[X|L3]) if ajout(L1,L2,L3).

/* Ecriture des éléments d'une liste à l'aide de
   la récursivité, si la liste est vide on n'écrit
   rien (arrêt), sinon on écrit la tête de liste et
   on recommence sur la queue de liste */

  écrireliste([]) if !.
  écrireliste([X|Q]) if writef("%4.0",X), écrireliste(Q).
```

■






```
VARIABLE XØ          VARIABLE YØ

CREATE LETTRE-N
6 C, 5 C, 4 C, 3 C, 2 C, 1 C, 2 C, 1 C,
2 C, 1 C, 2 C, 1 C, 2 C, 1 C, 2 C, 1 C,
2 C, 5 C, 5 C, 5 C, 6 C, 5 C, 5 C, 6 C,
5 C, 5 C, 6 C, 5 C, 5 C, 4 C, 2 C, 1 C,
1 C, 2 C, 1 C, 2 C, 1 C, 2 C, 1 C, 2 C,
1 C, 2 C, 2 C, 3 C, 4 C, 5 C, 6 C, 7 C,
Ø C,
```

Les variables XO et YO déterminent la position du point à afficher. Le mot ORIENTATION détermine le décalage à affecter aux codes du motif à tracer. Cette astuce permet de tracer le même motif avec huit orientations différentes. Exemple:

Enfin, le mot AFFICHER trace votre dessin. Pour exemple, LETTRE-N 20 20 AFFICHER trace la lettre N manuscrite à partir du point de coordonnées 20 20. 49 codes suffisent à définir notre lettre N et le dessin peut être tracé selon huit orientations différentes!.

```
SCR: 22
VARIABLE SENS

: ORIENTATION  ( n ---)
SENS ! ;

: AFFICHER ( adr XØ YØ ---)
YØ ! XØ !
BEGIN
  XØ @ YØ @ PSET
  DUP 1+ SWAP
  C@  DUP 1- SENS @ 1- +  8 MOD 1+
  CASE
    1 OF              -1 YØ +! ENDOF
    2 OF  1 XØ +!    -1 YØ +! ENDOF
    3 OF  1 XØ +!             ENDOF
    4 OF  1 XØ +!     1 YØ +! ENDOF
    5 OF              1 YØ +! ENDOF
    6 OF -1 XØ +!     1 YØ +! ENDOF
    7 OF -1 XØ +!             ENDOF
    8 OF -1 XØ +!    -1 YØ +! ENDOF
  ENDCASE
   Ø=
UNTIL
DROP ;
```

Ainsi, le codage de HUFFMAN permet de définir un alphabet comportant un nombre quelconque de caractères. Les caratères pourront être tracés dans n'importe quelle direction ce qui est pratique pour certains alphabets, dont l'alphabet arabe. Ce principe de codage autorise la superposition de caratères donc la ponctuation relativement complexe de l'arabe ou de tout autre alphabet.

FORTH EN ARABE

Voici une proposition tout à fait audacieuse: définir FORTH en arabe. L'idée est plus simple qu'il n'y parait au premier abord. L'affichage d'un mot en caratères latins ou arabes serait déterminée en fonction de l'état du bit non utilisé dans le nfa du mot: bit à 0, caratères latins, affichage conventionnel; bit à 1, caratères arabes, affichage en codage de FREEMAN, ainsi conserverait-on la mixité des alphabets. Les nombreuses fonctions vectorisées par F83 permettraient de générer une saisie clavier directement en caratères arabes avec exécution de tout mot saisi au clavier.

Dans le même ordre d'idée, un FORTH-GORBATCHEV (sans arrière pensée politique...) pourrait être envisagé. ■

suite de la page 15

assembleur écrit en langage FORTH peut sembler superflue. En effet, pourquoi réinventer une nouvelle syntaxe au lieu d'utiliser celle déjà existante dans un assembleur conventionnel et largement utilisée en programmation.

La conservation d'une syntaxe conventionnelle oblige à définir en FORTH un programme d'analyse syntaxique très encombrant. Il faut rechercher les séparateurs tels la virgule, le crochet carré ouvrant ou fermant, les éléments de chaîne à convertir en nombre, etc... bref des opérations délicates. Un tel analyseur syntaxique n'admettrait plus l'usage de mots FORTH et la définition de macro-instructions deviendrait impossible.

La solution retenue est inspirée des assembleurs implantés et utilisables en FORTH pour les systèmes équipés de micro-processeurs Z80, 8080, 8086, 6502, 68000, VAX ASSEMBLER, PDP-11 ASSEMBLER...

La syntaxe du code source défini à l'aide de cet assembleur est compatible avec celle définie dans la version eFORTH (de Frank HOOG Laboratory) et reprise par l'ensemble de la communauté FORTH internationale. Les extensions particulières sont définies dans les blocs 31 et 32. Une gestion minimale des erreurs est assurée en cas de défaut d'opérande ou de mode d'adressage erroné. Les nombreux commentaires inclus dans le programme source permettent à l'utilisateur de comprendre le fonctionnement de cet assembleur.

L'assembleur FORTH est constitué de 25 blocs, numérotés de 7 à 32. Dans le bloc no7, les mots DEFER et IS permettent la gestion de vecteurs telle qu'elle est définie par le FORTH 83-STANDARD. La compilation du bloc 8 est facultative. Son contenu, s'il est compilé, définit une suite de vecteurs dont le but est de permettre l'assemblage croisé, c'est à dire la génération d'un code objet pouvant être exécuté sans la cartouche FORTH. Cette extension reste à définir.

L'assembleur proprement dit commence au bloc 11. Toutes les définitions gérant cet assembleur sont utilisables sur d'autres systèmes équipés du micro-processeur 6809 (GOUPIL 2 et 3, TAVERNIER, VEGAS 6809, SQUALE, DRAGON 32 et 64 et tout système tournant sous FLEX ou OS9-6809). Il peut également être repris au sein d'un méta-compilateur FORTH équipant les systèmes IBM et compatibles (FORTH 83 de Laxen et Perry, FORTH LMI) pour générer un code objet qui sera tranféré et utilisé sur un système équipé d'un micro-processeur 6809.

Par convention, on désigne les différents éléments de la syntaxe du langage d'assemblage par les termes de labels d'opérateurs et d'opérandes. Dans un listing source écrit en assembleur conventionnel (celui disponible dans la cartouche ASSEMBLEUR pour TO7/70 et TO9 par exemple), chacun de ces éléments est utilisable dans une zone déterminée de l'écran. Ainsi, les colonnes 0 à 7 sont réservées aux labels et étiquettes; les colonnes 8 à 15 aux opérateurs (appelés aussi mnémoniques) ou aux directives d'assemblage et, éventuellement, aux macro-instructions; les colonnes 16 à 23 aux opérandes; les colonnes restantes contenant des commentaires.

suite page 18

Dans l'assembleur FORTH, on trouvera l'opérande, suivi éventuellement d'un indicateur de mode d'adressage, puis l'opérateur ou la macro-instruction. L'écriture de ces différents éléments se fait 'en ligne'. Il n'y a pas

On propose sous ce titre une série de petits programmes APL courts (tout au moins en APL) faciles, amusants et parfois utiles, et celà dans le but de familiariser le débutant avec le langage APL. Aujourd'hui:

LE CRIBLE D'ERATHOSTHENE

C'est un procédé qui permet de répertorier tous les nombres premiers inférieurs ou égaux à un nombre donné N.

Le principe est simple: on dresse un tableau de tous les nombres dans l'ordre croissant, à partir de 2 et on barre tous les multiples de ce nombre qui se trouvent dans le tableau. A la fin des opérations, il nbe reste plus que les nombres entiers.

Pratiquement, on partira d'un tableau T unidimensionnel, de N éléments égaux à 1, le rang de chaque élément du tableau est alors compris entre 1 et N inclus. Ceci s'écrit en APL:

```
T ← N ⍴ 1
```

(le ⍴ (rho) est en APL l'opérateur "DIMENSION").

Ensuite on mettra à zéro tous les éléments du tableau dont le rang est multiple de I (sauf I lui-même), I étant un indice qu'on fera varier de 2 à N.

Plutôt que de faire une boucle (toujours à éviter dans un langage interprété) pour calculer les multiples de I, on les calculera directement en sachant que leur nombre (I inclus) est la valeur entière de N divisée par I. Soit en APL:

```
L N ÷ I
```

Rappelons encore ici qu'une expression APL esr toujours évaluée de droite à gauche, sans priorité entre les opérateurs.

L'opération ⍳K (iota de K, K entier) génère les K premiers nombres entiers dans l'ordre croissant. Les multiples de I (I compris) situés dans le tableau seront générés par l'expression APL:

```
I x ⍳ L N ÷ I
```

Si l'on veut exclure I lui-même, il faut en "laisser tomber" le premier élément grâce à l'opérateur "DROP" symbolisé par ↓ . Finalement les indices des éléments à mettre à zéro seront donnés par l'expression APL:

```
1 ↓ I x ⍳ L N ÷ I
```

Ceci étant, on pourra écrire la fonction suivante:

```
[0]  Z←ERATO N
[1]  ⍝ *** Crible d'Eratosthene ***
[2]  T←N⍴I←⎕IO←1
[3]  LOOP:→(N<I←I+1)/END
[4]  →(T[I]=0)/LOOP
[5]  T[1↓I×⍳⌊N÷I]←0
[6]  →LOOP
[7]  END:Z←T/⍳N
```

La ligne [3] commence par le mot LOOP suivi de : ce qui signifie que ce mot est une étiquette servant à repérer la ligne.

Ensuite, sont réalisées successivement l'incrémentation de I, et lorsque I est devenu supérieur à N, le branchement vers l'étiquette END, ce qui fait sortir de la boucle.

Ce branchement est fait de manière classique en APL en écrivant:

→(condition) / étiquette

Nous n'en détaillerons pas la syntaxe maintenant, mais nous retiendrons le procédé.

La ligne [4] n'est pas indispensable, mais fait gagner du temps à l'exécution. Si I n'est pas premier (si T[I]=0), il est inutile de supprimer ses multiples du tableau: c'est déjà fait! On passe tout de suite à la valeur suivante de I en se branchant en début de boucle.

La ligne [5] est en fait la seule "vraie" ligne d'instruction du programme, les autres ne servant qu'à des initialisations, des incrémentations, des branchements... l'intendance. Elle affecte la valeur 0 à toutes les cases du tableau correspondant aux multiples de I. Celà se fait en une seule instruction, sans rajouter de boucle supplémentaire.

La ligne [6] nous renvoie en début de boucle.

La ligne [7] sert à "sortir" les nombres premiers qui sont en fait les indices des éléments du tableau qui sont égaux à 1. Arrêtons nous y un instant car elle utilise un opérateur très utile en APL, l'opérateur COMPRESSION / (remarque: la division c'est ÷). Le vecteur (la liste) des N premiers nombres entiers généré par ⍳N, est compressé par le vecteur T de même longueur que lui, mais composé de 0 et de 1. Seuls les éléments correspondants à des 1 sont conservés.

Comme un petit exemple vaut mieux qu'un long discours, voici ce qui se passe pour N=10:

N	1	2	3	4	5	6	7	8	9	10
T	1	1	1	0	1	0	1	0	0	0
Z	1	2	3		5		7			

Le débutant en APL pourra facilement éditer ce petit programme de sept petites lignes. A titre de comparaison, il faudrait de 25 à 30 lignes au minimum pour l'équivalent Basic ou Pascal (Ndlr: ..en Pascal, un programme est destiné à être compilé, donc le résultat ne s'exprime plus en nombre de lignes, idem pour FORTH dont le crible d'Eratosthène est un programme de test de performances - voir l'article concernant le NOVIX4000 dans le no32). ■

Suite de la page 17

d'emplacement défini pour les labels, les opérateurs et les opérandes. Seul compte l'ordre d'écriture de ceux-ci. Les commentaires sont écrit tels que définis dans le standard FORTH, c'est à dire entre parenthèses.

Les opérandes peuvent être des constantes ou des variables définies en FORTH, des nombres entiers, des labels pour les instructions de branchement. Toutes les bases numériques sont utilisables (oui, toutes, c'est à dire de 2 à 36 sans discontinuité). Exemple:

```
16384 # ldx
ou     HEX 4000 # ldx
ou     HEX 4000 CONSTANT STAD
```

...et plus loin...

```
STAD # ldx
```

Les éléments de l'instruction (opérande, opérateur, indicateur de mode d'adressage) seront séparés par au moins un espace et les instructions écrites sur une même ligne par trois espaces au moins, ceci afin d'accroître la lisibilité du programme. Exemple:

```
25 # lda   a,s ldx
```

Les opérateurs et indicateurs de mode d'adressage sont tous définis en caractères minuscules, évitant ainsi les confusions entre les nombres exprimés en base hexadécimale et certains opérateurs, tels DECB et decb par exemple.

L'ADRESSAGE INHERENT

L'adressage inhérent simple ne comporte aucun opérande et est utilisé par les instructions agissant directement sur les registres internes du 6809. Ces instructions sont définies dans le bloc 11. Exemple:

```
rola   decb   mul
```

En adressage inhérent paramétré, l'opérateur est précédé par un ou plusieurs mots chargés de contrôler un paramètre utilisé ensuite par les opérateurs exg, tfr, pulu, puls, pshu et pshs. Exemple:

```
a b exg
x d tfr
b x y pshs
```

→

Les noms donnés aux registres sont:

```
a b d   pour les accumulateurs
u s        les pointeurs de pile
x y        les registres d'index
pcr        le compteur programme
dpr        le registre 'page directe'
ccr        le registre de code condition
```

Dans le cas d'un empilage des registres, il ne faut pas empiler u ou s sur les piles respectives. Ainsi, la séquence 'u pulu' est erronnée (en fait, ce sera le registre s qui sera empilé).

Dans le cas de l'opérateur cwai, l'opérande représente une valeur litérale et utilisé comme les opérandes de l'adressage immédiat, ceci à l'exclusion de tout autre mode d'adressage. Exemple:

```
HEX FF # cwai.
```

LE MODE D'ADRESSAGE IMMEDIAT

L'opérande est suivi d'un indicateur de mode d'adressage représenté par le signe '#'. Exemple:

```
65 # lda  ( charge a avec valeur 65)
```

L'opérande est toujours un nombre entier, mais son intervalle de définition dépend du registre affecté. Pour un registre huit bits, cet intervalle est compris entre 0 et 255; pour un registre seize bits, il est compris entre 0 et 65535. Si l'opérande affecté à un registre huit bits est en dehors de cet intervalle, seule la partie de poids faible sera prise en compte. Ce cas ne produit pas de message d'erreur.

L'ADRESSAGE ETENDU

L'instruction comprend un opérande seize bits, suivi de l'opérateur. Exemple:

```
16384 lda  ( charge a avec contenu adr 16384)
```

En cas d'hésitation, vous pouvez forcer le mode d'adressage étendu en utilisant le signe ">". Exemple:

```
16384 > lda
```

L'adressage étendu admet un niveau d'indirection supplémentaire. Cette indirection est marquée par l'emploi du signe "]". Exemple:

```
HEX 2000 ] lda
```

équivalant à LDA[$2000] en assembleur conventionnel.

LE MODE D'ADRESSAGE DIRECT

L'instruction est composée de l'opérande, suivi du signe "<" dont le rôle est de forcer l'adressage direct, et de l'opérateur. Exemple:

```
HEX 602D < lda   et   HEX 2D < lda
```

sont équivalents, si le contenu de dpr a été initialisé avec l'octet de poids fort de l'opérande, ici 60 pour 602D en hexadécimal.

Cette initialisation est programmée par l'utilisateur comme suit:

```
HEX 60 # lda   a dpr tfr
```

Attention, l'assembleur FORTH ne passe pas automatiquement du mode d'adressage étendu au mode d'adressage direct dans le cas où la partie poids fort de l'adresse correspond au contenu du registre dpr.

LE MODE D'ADRESSAGE INDEXE

C'est le plus complexe des modes d'adressage. L'indexation fait appel à une base et un déplacement. La base peut être un des deux registres d'index x ou y, mais également un des pointeurs de pile u ou s, ou dans certains cas le compteur programme pcr.

Le déplacement peut être une valeur litérale nulle, ou exprimée sur cinq, huit ou seize bits. Ce déplacement peut aussi être désigné par le contenu d'un des accumulateurs a, b ou d. Enfin, la base peut subir une pré-décrémentation simple ou double, ou une post-incrémentation simple ou double. Dans certains cas, une indirection supplémentaire peut être sélectionnée.

Les indicateurs de mode d'adressage indexé avec déplacement constant sont définis dans le bloc 11 et sont les suivants:

```
,x ,y ,u ,s
```

et le déplacement est toujours indiqué, même dans le cas d'un déplacement nul. Exemples:

```
0 ,u ldd
6 ,u ldd
```

Les indicateurs du mode d'adressage indexé avec déplacement variable sont définis dans le bloc 16 et sont les suivants:

```
a,x  b,x  d,x  a,y  b,y  d,y
a,u  b,u  d,u  a,s  b,s  d,s
```

```
Exemple:             6 # lda   a,u ldx
est équivalent à     6 ,u ldx
```

Les indicateurs du mode d'adressage indexé avec pré-décrémentation et post-incrémentation sont définis dans le bloc 15 et sont les suivants:

```
,x+   ,x++  ,x-   ,--x
,y+   ,y++  ,-y   ,--y
,u+   ,u++  ,-u   ,--u
,s+   ,s++  ,-s   ,--s
```

```
Exemple:              ,x+ lda
  est équivalent     0 ,x lda   1 ,x leax
```

Un niveau d'indirection supplémentaire peut être sélectionné, sauf dans le cas d'une pré-décrémentation simple ou une post-incrémentation simple. Dans le cas du déplacement constant sur cinq bits, ce déplacement est convertit en déplacement huit bits. Exemple:

```
CODE CALL  ( adr --)  0 ,u ] jsr
NEXT   END-CODE
```

Dans le cas de l'adressage indexé utilisant le compteur programme comme base, il y a certaines restrictions. Le déplacement est précisé sous forme d'adresse, le mot 'pcr' convertissant cette adresse en valeur relative. Le déplacement huit ou seize bits est sélectionné automatiquement. Il n'y a pas d'indirection supplémentaire pour ce cas. Exemple:

```
DECIMAL  VARIABLE MASQUE  127 MASQUE !
```

et plus loin, dans une définition:

```
MASQUE ,pcr adda
```

LES BRANCHEMENTS

Ils sont de deux sortes, les branchements inconditionnels absolus et les branchements conditionnels relatifs.

Les branchements inconditionnels absolus correspondent à des branchements réalisés vers une adresse exprimée littéralement. Les deux branchements possibles dans ce cas sont jmp et jsr. Ces deux mots sont utilisables en adressage étendu, direct ou indexé et admettent un niveau d'indirection supplémentaire en adressage étendu ou indexé. Exemples:

```
HEX 0099 jmp ( adr étendu)

HEX 0099 # ldd   d pshu
0 ,u jmp    ( adr indexé, dépl. nul)

VARIABLE SUITE  0099 SUITE !
...et plus loin, dans une définition...
SUITE # ldd   d pshu
0 ,u ] jmp  ( adr indexé, dépl. nul avec indir.)
```

Les branchements conditionnels relatifs, définis dans le bloc 28, sont précédés d'une adresse absolue. La conversion en valeur relative, pour exécuter un branchement court ou long, est automatique. Seuls les branchements conditionnels vers l'arrière sont possibles. Exemple:

```
16384 CONSTANT STAD
 CODE RAZ  ( —)
 x pshs   0 # lda
 LABEL RETOUR    ,x+ sta
   STAD 8000 + # cmpx    RETOUR BNE
 x puls   NEXT
 FORGET-LABEL RETOUR
 END-CODE
```

Les deux instructions de branchement relatif bra et bsr sont inconditionnelles.

LABELS ET STRUCTURES DE CONTROLE

L'ensemble des mnémoniques permettant l'assemblage de branchements conditionnels ne peuvent être utilisés que pour les branchements vers l'arrière, donc vers une partie de définition déjà assemblée. Le branchement peut être réalisé vers une partie de définition différente de la définition en cours d'assemblage. L'adresse de branchement peut être le début de la zone paramétrique d'une autre définition. Exemple:

```
CODE <mot1>
... définition ... END-CODE
CODE <mot2>
... définition ...
' <mot1> bne ... END-CODE
```

En dehors de ce cas, on ne peut faire appel qu'aux labels. Les labels sont des pseudo-constantes dont les en-têtes sont définis de manière transitoire dans une zone réservée du dictionnaire et accessible à partir du vocabulaire ASSEMBLER exclusivement. Un label se définit dans une séquence en cours d'assemblage comme suit:

```
LABEL <mot>
```

le nombre de labels définissables simultanément est limité et varie en fonction de la taille des noms utilisés. Comptez une quinzaine de labels pour des noms de 6 à 10 caractères de longueur. Cependant, on peut récupérer l'espace alloué aux labels après utilisation, en exécutant la séquence:

```
FORGET-LABEL <mot>
```

Tout label défini après <mot> sera oublié. L'effacement du premier label défini détruit tous les autres labels. Pour exemple, voir la définition de RAZ citée précédemment.

Afin de limiter la prolifération des labels, on peut faire appel aux structures de contrôle. Ces structures, similaires à celles utilisables dans une définition de haut niveau, permettent de réaliser des branchements vers l'avant. Les structures de contrôle disponibles dans le vocabulaire d'assemblage sont les suivantes:

```
if ... then
if ... else ... then
begin ... until
begin ... while ... repeat
```

Leur usage accroit considérablement la lisibilité du programme. Les conditions d'exécution du branchement sont placées avant le mot if pour les deux premières structures citées ci-dessus; avant le mot until pour la troisième structure; avant le mot while pour la dernière structure. Ces conditions sont définies dans le bloc 29. Les équivalences avec les instructions de branchement sont cc pour bcc, cs pour bcs, eq pour beq, etc...

Important: les mots if, then, else, begin, while, repeat et until bien qu'ayant leurs homonymes définis en caractères majuscules dans le vocabulaire FORTH, ne sont utilisables qu'au sein d'une séquence en cours d'assemblage. Une séquence du type 'eq IF..then' est erronnée. Sous ASSEMBLER, il n'y a pas de vérification de cohérence de la structure. Une séquence du type 'begin .. if .. until .. then' vous amènera quelques surprises à l'exécution bien qu'aucune erreur ne soit signalée à l'assemblage. En effet, on peut vouloir "déstructurer" volontairement deux structures de contrôle imbriquées. Cas de deux structures de contrôle imbriquées et structurées:

```
CODE <mot> ...
cond if
  cond if .. then
then ..
```

La "destructuration" peut être obtenue en inversant les adresses de référence des branchements avant résolution:

```
CODE <mot> ...
 cond if
   cond if .. SWAP
 then
   then ..
```

ce qui a pour effet d'assembler des branchements croisés. Mais en règle générale, il est conseillé de respecter la structuration des séquences délimitées par les structures de contrôle.

LES MACRO-INSTRUCTIONS

Sous ce nom barbare se cache une des plus puissantes options d'assemblage. Précisons un peu ce qu'est une macro-instruction pour l'assembleur FORTH.

Lors d'une séquence d'assemblage, il arrive fréquemment qu'une même suite d'instructions apparaisse au cours du programme. Il peut donc être intéressant de définir un mot, qui, lors de son exécution, sera chargé d'assembler toute une suite de codes.

Voyons un exemple très simple et fréquent; la paire de registres accumulateurs 8 bits a et b peut être considérée comme un seul registre accumulateur 16 bits nommé d, les registres a et b représentant respectivement les partie poids fort et poids faible du registre d. Or, s'il existe des instructions permettant le décalage du contenu de a et b, il n'existe rien de tel pour le registre d. Si on veut décaler le contenu du registre d, il faut faire appel à une suite d'instructions agissant sur le contenu des registres a et b séparément. Ainsi, pour provoquer un décalage logique à gauche du contenu du registre d, il faut taper:

```
aslb  rola
```

La multiplication par huit, en valeur non signée, du contenu de d peut se définir:

```
aslb rola aslb rola aslb rola
```

Ce qui est un peu répétitif. La définition d'une macro-instruction, en assembleur conventionnel sur les systèmes disposant de cette fonction (donc pas sur les cartouches ASSEMBLEUR pour THOMSON TO7/70/TO9) se définit ainsi:

```
LSLD    MACRO
        ASLB
        ROLA
        ENDM       Pour ENDMacro
```

Et en assembleur FORTH, une macro-instruction se définit dans un mot de type "deux-points":

```
: lsld          ASSEMBLER
aslb   rola  FORTH  ;
```

Ce qui permet de réécrire la multiplication du contenu de d par huit:

```
lsld  lsld  lsld
```

En somme, créer une macro-instruction, c'est créer un nouveau mnémonique. A titre d'exemple, voici une série de macro-instructions fréquemment utilisées.

```
HEX
: aba    ( A:=A+B)
ASSEMBLER  b pshs    ,s+ adda    FORTH ;

: lsrd  ( D:=D/2)
ASSEMBLER  lsra   rord    FORTH  ;
```

Suite au prochain numéro